2008

Counting and sampling paths in graphs

T. Ryan Hoens

Counting and Sampling Paths in Graphs

by

T. Ryan Hoens

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Supervised by

Ivona Bezáková
Department of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York
August 2008

Approved By:

Ivona Bezáková
Primary Advisor

Stanisław P. Radziszowski
Reader

Piotr Faliszewski
Observer

Thesis Release Permission Form

Rochester Institute of Technology

B. Thomas Golisano College of Computing and Information Sciences

Title: Counting and Sampling Paths in Graphs

I, T. Ryan Hoens, hereby grant permission to the Wallace Memorial Library reproduce my thesis in whole or part.

T. Ryan Hoens

Date

Acknowledgments

During the course of my time here at R.I.T. I have had the opportunity to meet many people who have made my time very productive.

I would first like to thank my advisor, Professor Ivona Bezáková for all of her help and guidance during the process of writing my thesis. Without her continued support and guidance, none of this would have been possible.

A special thank you goes to Professor Stanisław Radziszowski for being a great source of advice and entertainment throughout the process. His general outlook on the process and good nature made the time pass more easily than it might have.

I would also like to thank Piotr Faliszewski for taking time out of his busy schedule to be my observer.

While this thesis signals the end of my time here at R.I.T., I would like to thank the people who helped me on the path of continuing my education, Edith Hemaspaandra, Chris Homan, and James Kwon. Without their support and encouragement I am not sure where I would be today. In addition I would like to thank my other professors at R.I.T. for making my stay here so enjoyable.

Finally I would like to thank William Rummler for lending me a helping hand when needed, providing me with a great sounding board when I needed to talk things out.

Abstract

Since Euler began studying paths in graphs, graph theory has become an important branch of mathematics. With all of the research into graph theoretic problems, however, counting – exactly or approximately – the number of simple paths in finite graphs has been neglected.

This thesis investigates an approximation technique known as *Markov chain Monte Carlo* (MCMC) for the specific purpose of approximating the number of simple paths in graphs. Due to the paucity of research into the subject, the thesis will make the conjecture that this cannot be done exactly in an efficient manner (assuming that the longstanding conjecture $P \neq NP$ holds).

To this end, the thesis focuses on the relationship between counting and sampling in both weighted and unweighted complete graphs, trees, and directed acyclic graphs (DAGs). This includes both positive and negative results for sampling, as well as demonstrating how counting and sampling are intimately related problems.

Contents

Chapter 1

Introduction

The Markov chain based approach to counting through sampling is a well studied approach with many diverse applications ([2][3][7][8][15]). While most sampling approaches attempt to sample uniformly from a set (and thus the corresponding counting problem is estimating the cardinality of the set), there are cases where one does not wish to sample uniformly. In some systems with applications to physics, for instance, configurations occur proportional to their *energy* (or in computer science, their *weight*). In these instances the counting problem does not compute the number of configurations, but rather the sum of their energies (in physics this is known as computing the *partition function*). In this thesis the Markov chain Monte Carlo (MCMC) technique is used to calculate the partition function for the number of paths in both unweighed (uniform sampling) and weighted (sampling from a non uniform distribution) graphs.

Sampling paths in arbitrary graphs has its use in modeling real world phenomena. While in this context it is often called sampling trajectories (*e.g.* [6][9][10]), the basic idea is the same. In each of these scenarios the underlying graph must first be inferred (or is known to be some network), and then paths are sampled to determine either likely behavior, or network usage.

To model these scenarios by the weighted complete graph, we would take the graph which they generated, and any missing edge would be added with a very low weight to discourage its use. In this way we could sample the trajectories which they desire.

In the thesis sample proofs are also provided to demonstrate the MCMC technique before

moving to the main results. The purpose of the sample proofs is not to show a new result for an unsolved problem, but rather to demonstrate the key concepts. To this end, the example chosen has a closed form solution so certain aspects of the proofs are more obvious, yet still demonstrate the complexities involved.

The example proofs and the results of this thesis share their need for two proofs to determine the existence of a fully polynomial randomized approximation scheme, or FPRAS (more formally defined in (2.3.2)). Namely, proving a reduction from *approximate counting* to *almost uniform sampling* in polynomial time, and a proof of a polynomial run time bound on the mixing time which defines approximately how long the Markov chain must run to reach the desired distribution (defined in (2.2.9)). When combined, these results show that given a fully polynomial almost uniform sampler, or FPAUS (defined in (2.4)), an FPRAS also exist.

The main results presented are the existence of an FPRAS for the weighted and unweighted complete graph (under some assumptions), as well as the unweighted tree. We also present a proof that the Markov chain defined for the unweighted tree mixes torpidly for the weighted tree. Finally, an algorithm is presented to compute the partition function for weighted trees and weighted DAGs. Given this counting algorithm, we then show how it can be used to sample paths proportional to their weight.

Chapter 2

Notation and Definitions

This section outlines important definitions which are used throughout the thesis. The two main areas which discussed are graph theory and Markov chains. For a more detailed look at graph theory, West ([29]) provides a good reference. For Markov chains, more detailed analysis and proofs can be found in Ross ([23]) and Häggström ([11]).

2.1 Graph Notation

A graph is an ordered pair of sets $G = (V, E)$, where V is a set of elements (called vertices), and E is a set of pairs of elements from the set V (called edges). In an undirected graph the pair of elements that make up an edge are considered unordered, while in a directed graph the pairs of elements are considered ordered. That is, the edge represented by the pair (v_1, v_2) is the same as that represented by the pair (v_2, v_1) in an undirected graph, but different in a directed graph. For the remainder of this thesis, we will refer to undirected graphs simply as graphs, and directed graphs as digraphs.

A cycle is a sequence of distinct vertices $(v_1, v_2, \ldots, v_k, v_1)$ such that for all $1 \leq i \leq k$, $(v_i, v_{i+1}) \in E$, and for all $i \neq j$, $v_i \neq v_j$. That is, one can walk along the sequence of vertices in the graph starting at v_1, and follow the path back to the starting vertex.

The three types of graphs discussed in this thesis are: *complete graphs*, *trees*, and *directed acyclic graphs* (DAGs). A *complete graph* is a graph in which all vertices in V are connected, i.e. $\forall v_1 \neq v_2 \in V$, $(v_1, v_2) \in E$. A tree is a graph with n vertices and $n - 1$

edges which is connected. A DAG is a directed graph (i.e. a graph with directed edges) which has no cycles.

A *simple path* (*self avoiding walk*) p in a graph $G = (V, E)$ is a sequence of distinct vertices, $p = (v_1 v_2 \dots v_k)$, such that for $0 < i < k, (v_i, v_{i+1}) \in E$. That is, it is a path through the graph such that no vertex occurs more than once As p denotes the (ordered) sequence of vertices, let $e(p)$ denote the (ordered) sequence of edges in p. As is common in graph theory, for the remainder of this thesis we will refer to simple paths as paths. The length of a path (denoted $|p|$) is the number of vertices in the path, where the path on 0 vertices is called the *empty path*. Finally, we use $\mathcal{P}(G)$ as the set of all paths in a graph, including the empty path.

A weighted graph $G = (V, E)$ is a graph with a function $w : E \to \mathcal{R}^+$. That is, it has a function which maps each edge to a nonnegative real number (or weight). A path p in a weighted graph G also has a weight (denoted $w(p)$). The weight[1] of p is defined as:

$$w(p) = \prod_{e \in e(p)} w(e).$$

(Note that the weight of paths without any edges is the empty product which is 1). That is, it is the product of the edge weights that compose the path. Given a subset $S \subseteq \mathcal{P}(G)$, $w(S)$ is defined as:

$$w(S) = \sum_{s \in S} w(s),$$

or the sum of all of the weights of the elements in S.

2.2 Markov Chain

The main mathematical construct behind the algorithms in this thesis is the Markov chain. This section is devoted to the definition of a Markov chain which will be used throughout

[1] Note that while weights are often defined as the sum of the edges, here the weight is defined as the product as is standard and more meaningful in many applications in physics.

the thesis. Since the approach taken requires the size of the state space to be the quantity estimated, we will only concern ourselves with Markov chains with finite state space.

2.2.1 Definition

A discrete time Markov chain (DTMC) is a discrete time stochastic process with the Markov property. A discrete time stochastic (random) process is a set of random variables $\{X_n : n > 0\}$, where X_n is a random variable over the state space Ω. The Markov property states that, given the present state, the conditional probability of any future state is independent from any state before the present. Mathematically:

$$P(X_{n+1} = i | X_n = j_n, X_{n-1} = j_{n-1}, \ldots, X_0 = j_0) = P(X_{n+1} = i | X_n = j_n)$$

We can then define p_{ij} to be the probability that the Markov chain, when in state i, will transit to state j one time step later. We also define p_{ij}^n to be the probability that the Markov chain, when in state i, will be in state j n time steps later. Since probabilities must be nonnegative values, $p_{ij} \geq 0$. We also know that if a Markov chain is in state i at time n, it must be in some state at time $n + 1$. That is, the Markov chain will not leave the state space in the next time step, and therefore, $\forall i \in \Omega, \sum_{j \in \Omega} p_{ij} = 1$.

We can then define P (known as the one step *transition matrix* of the Markov chain) as:

$$P = \begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdots \\ p_{10} & p_{11} & p_{12} & \cdots \\ p_{20} & p_{21} & p_{22} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Note that given this definition, $\forall x \in \Omega$, $P(x, \cdot)$ (i.e. row x) is a probability distribution.

While P denotes the probability of going from state x to state y in one step, P^n denotes the probability of going from state x to state y in n steps. Thus P^n is called the n step transition matrix.

2.2.2 Lazy Markov Chain

A Markov chain is considered *lazy* if, for all $x \in \Omega$, $P(x, x) \geq 1/2$. To create a lazy Markov chain from an arbitrary Markov chain, it suffices to divide each element in P by 2, and then add probability $1/2$ to each element along the diagonal (equivalent to adding $I \cdot 1/2$ where I is the identity matrix).

2.2.3 Reducibility

A state j is *accessible* from state i if, when in state i there exists an n such that:

$$\Pr\left(X_n = j | X_0 = i\right) > 0$$

That is, there is a way of transitioning through the Markov chain from i to j. Note that by setting $n = 0$, any state i is accessible from itself.

States i and j *communicate* if i is accessible from j and j is accessible from i. A set of states S is a *communicating class* if every pair of states in C communicate, and no state in C communicates with a state not in C.

A Markov chain is *irreducible* if its entire state space is a communicating class.

2.2.4 Periodicity

A state i has a period of k if, given that the Markov chain is in state i at some time n, it can only return to state i in time steps which are multiples of k. A state with a period of 1 is said to be aperiodic.

2.2.5 Recurrence

A state i is recurrent if the probability of returning to state i after visiting state i is positive. A state is *positive recurrent* if it is recurrent and if the expected value of the number of steps it takes to return to state i is finite. Otherwise, the state is called *null recurrent*. Since the

Markov chains discussed in this thesis all have finite state space, it is useful to note that any recurrent state in a finite Markov chain is positive recurrent.

2.2.6 Ergodicity

An ergodic state i is one which is both aperiodic and positive recurrent. An ergodic Markov chain is one for which all of its states are ergodic.

2.2.7 Stationary Distribution

A stationary distribution (sometimes called the limiting probabilities) of a Markov chain is a probability distribution π which measures the long term probability of any state j. Formally,

$$\pi_j = \sum_{i \in \Omega} \pi_i p_{ij}$$

and:

$$\sum_{i \in \Omega} \pi_i = 1.$$

From Ross ([23]) we have the following theorem about the stationary distributions.

Theorem 2.2.1 (Ross Theorem 4.1)**.** *For an irreducible ergodic Markov chain* $\lim_{n\to\infty} P_{ij}^n$ *exists and is independent of* i*. Furthermore, letting*

$$\pi_j = \lim_{n\to\infty} P_{ij}^n, \qquad (j \geq 0)$$

then $\pi_1, \pi_2, \ldots, \pi_{|\Omega|}$ *is the unique nonnegative solution of*

$$\pi_j = \sum_{i \in \Omega} \pi_i P_{ij}, \qquad (j \geq 0)$$

$$\sum_{j \in \Omega} \pi_j = 1$$

It is important to note that if the transition matrix of a Markov chain is symmetric, then the stationary distribution is uniform.

2.2.8 Reversibility

Time reversibility (sometimes called detailed balance) is a property of a Markov chain which states, given $i, j \in \Omega$,

$$\pi_i p_{ij} = \pi_j p_{ji}.$$

It is easy to see that for an ergodic Markov chain the π satisfying the detailed balance condition is also the stationary distribution of the Markov chain.

2.2.9 Mixing time of a Markov chain

Intuitively, the mixing time of a Markov chain is the minimum number of steps that it takes before the distribution of states is within some ϵ of the stationary distribution. In order to measure the distance from the stationary distribution, we use the *total variation distance*. Given two probability distributions π and μ, the total variation distance between the two distributions is defined as:

$$d_{TV}(\mu, \pi) = \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \pi(x)|$$

Therefore, mathematically, the mixing time of a Markov chain with stationary distribution π is defined as:

$$\tau_x(\epsilon) = \min\{n : d_{TV}(P^n(x, \cdot), \pi) \leq \epsilon\}$$

It is important to note that $P^i(x, \cdot)$ is a probability distribution over Ω corresponding to the Markov chain starting from x and making i steps. This is due to the fact that $P^i(x, \cdot)$ merely refers to row x in the i step transition matrix. In fact $\lim_{n \to \infty} P^n(x, \cdot)$ is exactly the stationary distribution π.

While this definition of the mixing time depends on the initial state x, the definition of the stationary distribution does not depend on the initial state. We therefore define the mixing time of the Markov chain when a mixing time irrespective of the initial state is desired:

$$\tau(\epsilon) = \max_{x \in \Omega} \tau_x(\epsilon)$$

A Markov chain is said to be *rapidly mixing* if $\tau(\epsilon)$ is bounded by a polynomial in the size of the input, and $\log(1/\epsilon)$. Conversely, a Markov chain is said to be *torpidly mixing* if $\tau(\epsilon)$ is bounded by an exponential (or worse) in the size of the input, and $\log(1/\epsilon)$.

In section (3.1.2) we will present two techniques used to bound the mixing time of a Markov chain.

2.3 Complexity and Approximation Schemes

In computer science, the determining factor of the efficiency of an algorithm is its running time, normally expressed in $\mathcal{O}$ (big-Oh) notation. $\mathcal{O}$ running times which are polynomial in all inputs are considered *efficient*, while those which run in time exponential (or worse) on any input are generally considered *inefficient*.

For some problems only inefficient algorithms are expected; in these cases we look for approximation schemes which are efficient. If, given an instance x, the approximation can be made accurate to within some arbitrary $\epsilon > 0$ (that is, the result obtained is within an $e^{\pm\epsilon}$ factor from the desired answer), the approximation is called an $e^{\pm\epsilon}$-approximation[2]. If the algorithm runs in time polynomial in $|x|$ and ϵ^{-1}, it is called a *fully polynomial-time approximation scheme* (FPTAS). If, however, the algorithm runs in time polynomial in $|x|$, but exponential in ϵ^{-1}, it is called a *polynomial-time approximation scheme* (PTAS). A more detailed analysis of approximation algorithms can be found in [13][27].

In some cases not even an (F)PTAS is known. In these cases *randomized approximation schemes* (RAS) might yield an effective approximation algorithm within some probabilistic (as well as ϵ) bound. The following sections defining RAS will closely follow lecture notes by Vigoda ([28]), which summarize the relevant sections of Jerrum ([15]).

2.3.1 Chernoff bound

In the following sections it will be useful to have the following Chernoff inequality:

[2]Note that by the Taylor series expansion this is essentially equivalent to a $(1 \pm \epsilon)$–approximation

Theorem 2.3.1 (Chernoff Inequality)**.** *Let $X_1, \ldots, X_n$ be independent, identically distributed (iid) {0,1}-random variables where $p = \mathcal{E}(X_i)$[3]. Then:*

$$\forall \epsilon \leq 1, \textit{Pr}\left(\left|\sum_{i=1}^{n} X_i - pn\right| > \epsilon pn\right) \leq 2e^{-\epsilon^2 pn/3}$$

This is one of the simpler versions of the Chernoff bounds, however it is enough for the purpose of this thesis. For a more detailed overview of Chernoff's inequalities see [1][4][14][20].

2.3.2 Randomized Approximation Schemes

A randomized approximation scheme aims to estimate a function f on input x. Given an input x, an approximation parameter $0 < \epsilon \leq 1$, and a confidence parameter $0 < \delta < 1/2$, the randomized approximation scheme g produces a random variable X such that:

$$\Pr\left(e^{-\epsilon} f(x) \leq X \leq e^{\epsilon} f(x)\right) \geq 1 - \delta$$

(Note, this is exactly the definition used by Jerrum in [15], and is used for the same reasons referenced therein.) A randomized approximation scheme which runs in time polynomial in $|x|$, ϵ^{-1}, and $\log(1/\delta)$ is said to be a *fully polynomial randomized approximation scheme* (FPRAS). For a more detailed overview on (randomized) approximation schemes see [13][15].

It is interesting to note that if the above holds for $\delta = 1/4$, then one can boost the error rate to an arbitrarily small δ'. The algorithm is defined as follows:

Choosing y to be the median results in the desired probabilities. This can be seen by defining:[4]

$$X_i = \begin{cases} 1 & \text{if } y_i \in e^{\pm\epsilon} f(x) \\ 0 & \text{otherwise} \end{cases}$$

[3] *Note that we use $\mathcal{E}$ to denote the expected value*

[4] Note that $x \in e^{\pm a}$ (or $\notin$) means x is (not) in the interval $[e^{-a}, e^{a}]$

BOOST-ERROR-RATE(g, δ')
1 Let $k = \lceil 16 \log(2/\delta') \rceil$.
2 Run k trials of the FPRAS g with error probability $1/4$, yielding $y_1, \ldots, y_k$.
3 Let y be the median of the k values
4 **return** x

Algorithm 1: *Algorithm to boost the error rate of of an FPRAS.*

Then by our error bounds we have that $\mathcal{E}(\sum X_i) \geq 3k/4$. Thus:

$$\begin{aligned}
\Pr\left(m \notin e^{\pm\epsilon} f(x)\right) &\leq \Pr\left(\sum X_i < k/2\right) \\
&\leq \Pr\left(\left|\sum X_i - \mathcal{E}\left(\sum X_i\right)\right| > k/4\right) \\
&\leq 2e^{-k^2/16k} && \text{(by Chernoff's inequality)} \\
&\leq \delta'
\end{aligned}$$

With this in mind, this report will only be concerned with FPRAS which have $\delta = 1/4$, and thus our requirement for an FPRAS is:

$$\Pr\left(e^{-\epsilon} f(x) \leq X \leq e^{\epsilon} f(x)\right) \geq 3/4. \tag{2.1}$$

Note that this is not a major restriction, as it will only add at most a logarithmic factor, as the above algorithm makes $\lceil 16 \log(2/\delta') \rceil$ iterations, which is then multiplied by the running time of the FPRAS.

2.4 Almost Uniform Samplers

Let Σ be the set of all instances of a problem P. Given an instance $x \in \Sigma$, define Ω_x to be the set of solutions of x and π be a distribution on Ω_x. We then want to sample from π, but sometimes this does not appear feasible. In such cases we attempt to devise an almost uniform sampler g which outputs solutions to x from a distribution μ. In order to measure the accuracy of g we use the total variation distance (as defined previously) between

the desired distribution π, and μ. If g generates samples from a distribution μ such that $d_{TV}(\mu, \pi) \leq \delta$, then g is called an *almost uniform sampler*. If an almost uniform sampler runs in time polynomial in $|x|$ and $\log(1/\delta)$, the sampler is called a *fully polynomial almost uniform sampler* (FPAUS).

Chapter 3

Background and Prior Work

While the MCMC technique has not been directly applied to counting the number of paths in finite graphs, the technique has seen usage in many other areas. Most closely related of which being Dana Randall's work on counting and sampling paths in the infinite lattice ([21]).

Due to the dependence of this thesis on previous theoretical work on the MCMC technique, a brief summary of the relevant work will be provided. Examples will also be provided to demonstrate some of the relevant techniques.

3.1 Theoretical Basis

While a complete introduction to the theory of MCMC is beyond the scope of this thesis, this section provides an overview of some of the major ideas. For a more in depth review Jerrum ([15]) and Sinclair ([25]) provide good references.

3.1.1 Reducing Approximate Counting to Almost Uniform Sampling

In order for the MCMC technique to be applicable to counting, a result showing the link between approximate counting and almost uniform sampling is necessary. This was accomplished by Jerrum, Valiant, and Vazirani ([17]) who proved that any so-called self-reducible problem which allows an almost uniform sampler can be approximately counted.

This work was subsequently summarized by Jerrum ([15]) for the specific case of matchings in a graph. In order to take advantage of this result in this thesis, however, one must demonstrate that each individual problem is self-reducible.

To denote a reduction, we use the symbol $\leq$. Thus, given two problems A and B, $A \leq B$ means that given an algorithm to solve B, a solution to A follows immediately.

3.1.2 Bounding the Mixing Time

The next two important results give a bound on the mixing time of a Markov chain. Before continuing it is important to note that a Markov chain can be viewed as a graph, where the set of vertices are defined by Ω, and the set of edges are defined by the transition matrix P.

In the first technique, called the *canonical paths* technique, one attempts to show that there exists a canonical path between every pair of states $x, y \in \Omega$ using only the transitions of the Markov chain, such that no edge is *overloaded* (i.e. which is depended upon by too many such pairs x and y). In order to ensure that no overloading occurs, we must measure the *congestion* of each edge of the Markov chain and prove that no bottleneck exists, thus implying that Markov chain is rapidly mixing.

The second technique, called *conductance* uses the idea of a minimum cut. More specifically, if there exists a reasonably sized portion of the states such that the Markov chain has a low probability of leaving, then it is not rapidly mixing. This is due to the fact that the set will be sampled from with higher probability than it contributes to the distribution unless more steps are taken to ensure a probability of leaving.

Canonical Path Technique

Given a Markov chain with state space Ω, two states are *adjacent* if the probability of transitioning between the two is non-zero. For all pairs of states $x, y \in \Omega$, define the canonical path $\gamma_{xy} = (z_1 z_2 \ldots z_k)$ (where $z_1 = x$ and $z_k = y$) as the path from x to y going through adjacent states in the Markov chain. Furthermore, we define: $\Gamma = \{\gamma_{xy} | x, y \in \Omega\}$ as the set of all canonical paths, and $\Gamma_{uv} = \{\gamma_{xy} | x, y \in \Omega, \text{ and } (u, v) \in \gamma_{xy}\}$. With this, we

define the congestion $\overline{\rho}$ of a Markov chain as:

$$\overline{\rho}(\Gamma) = \max_{(u,v):P(u,v)>0} \left\{ \frac{1}{\pi(u)P(u,v)} \sum_{x,y:\gamma_{xy}\in\Gamma_{uv}} \pi(x)\pi(y)|\gamma_{xy}| \right\}$$

where $|\gamma_{xy}|$ is the length of the path.

In order for this approach to be successful, $\overline{\rho}$ should imply something about the mixing time of the Markov chain. Jerrum goes through the derivation of this in [15], eventually proving:

Theorem 3.1.1 (Jerrum [15])**.** *The mixing time* $(\tau_x(\epsilon))$ *of a lazy Markov chain is bounded by:*

$$\tau_x(\epsilon) \leq 2\overline{\rho}\left(2\ln\epsilon^{-1} + \ln\pi(x)^{-1}\right),$$

where $\overline{\rho} = \overline{\rho}(\Gamma)$ *is the congestion (defined previously) with respect to any set of canonical paths* Γ

This technique will therefore provide a good upper bound on the number of steps the Markov chain requires to sample from (close to) the stationary distribution.

Conductance Technique

While the canonical path technique is used to give an upper bound for the mixing time, a popular technique for giving a lower bound is called *conductance*. This technique, introduced by Jerrum and Sinclair ([26]), relates the mixing time of a Markov chain to the probability of leaving a set of states in the Markov chain.

If $S \subset \Omega$, then we define the probability that the Markov chain leaves set S given that the Markov chain is in a state in S as: $\partial S = \sum_{x\in S, y\notin S} \pi(x)P(x,y)$. Given this, we can then define:

$$\Phi_S = \frac{\partial S}{\pi(S)} = \frac{\sum_{x\in S,y\notin S}\pi(x)P(x,y)}{\pi(S)}.$$

The conductance is then defined as:

$$\Phi = \min_{S\subset\Omega:\pi(S)\leq 1/2} \Phi_S.$$

Given this definition of conductance, the following is a result from Jerrum and Sinclair ([26]):

Theorem 3.1.2 (Jerrum and Sinclair [26])**.** *For any Markov chain with conductance* Φ*,*

$$\frac{\Phi^2}{2} \leq Gap(P) \leq 2\Phi.$$

Where $Gap(P)$ *is the spectral gap of the transition matrix* P*.*

In order to use this to bound the mixing time, we refer to Randall's overview (in [22]). Therein she concisely demonstrates how to use the concept of the Markov chain's *spectral gap* to bound its mixing time. That is, letting $\lambda_1, \lambda_2, \ldots, \lambda_{|\Omega|}$ be the eigenvalues of the Markov chain's transition matrix P such that $1 = \lambda_1 > |\lambda_2| \geq \ldots \geq |\lambda_{|\Omega|}|$, the spectral gap is defined as $Gap(P) = 1 - |\lambda_2|$. Using this definition, we have from [17]:

Theorem 3.1.3. *Let* $\pi_* = \min_{x \in \Omega} \pi(x)$*. For all* $\epsilon > 0$*, we have:*

$$\tau(\epsilon) \leq \frac{1}{Gap(P)} \log\left(\frac{1}{\pi_* \epsilon}\right)$$

and

$$\tau(\epsilon) \geq \frac{1}{2Gap(P)} \log\left(\frac{1}{2\epsilon}\right).$$

Given this bound on $Gap(P)$, we can now combine Theorem 3.1.2 and Theorem 3.1.3 to realize the following bound:

$$\tau(\epsilon) \geq \frac{1}{2\Phi} \log \frac{1}{2\epsilon}$$

Using Theorem 3.1.2 it is easy to see that if the conductance, Φ, of a Markov chain is inverse exponential, the Markov chain will be torpidly mixing.

3.1.3 Metropolis-Hastings Algorithm

The Metropolis-Hastings Algorithm provides a tool for defining a Markov chain with a specific stationary distribution (*e.g.* non-uniform). The algorithm achieves this by limiting the probability that the Markov chain makes a *bad* choice. Given two states $x, y \in \Omega$ and

a stationary distribution π, a bad decision is one for which $\pi(x) > \pi(y)$, and the Markov chain transitions from x to y. That is, a bad decision occurs when the MC transitions from a state with higher likelihood to one with lower likelihood.

In order to discourage this behavior, the Metropolis-Hastings algorithm slightly modifies how transitions are made. Originally, if a Markov chain in state x generates state y as the next state, the MC immediately transitions to state y. The Metropolis-Hastings algorithm, however, defines that transition only to be taken with probability $p = \min\{1, \pi(y)/\pi(x)\}$, staying at x otherwise.

For more information on the Metropolis-Hastings Algorithm, [5] gives a good overview. The original algorithm is defined in [19], with extensions in [12].

3.2 Prior Work

The use of Markov chains to approximately count the size of some combinatorial structure has become more prevalent recently. It has also been used in quite diverse applications. The seminal paper in this area is by Jerrum, Valiant, and Vazirani ([17]). Building on this result, Dyer, Frieze, and Jerrum ([8]) used the technique to approximate the number of independent sets in sparse graphs. Jerrum, Sinclair, and Vigoda ([16]) approximated the permanent of a non-negative matrix; this result was later improved by Bezáková, Štefankovič, Vazirani, and Vigoda ([3]). The wide area of applications shows the promise of this approach, and thus its choice for this thesis.

Prior work in the area of counting paths in a (specific type of) graph was carried out by Randall in [21]. Randall's work differs from this thesis in that she looked exclusively at paths of length n in a specific type of graph called an *infinite lattice* due to its applications in chemical physics. Her work also used the Markov chain based approach, however there are key differences in the construction of the Markov chain used due to the difference between a finite graph, and an infinite graph. More importantly, the results she obtained were promising, and show that the Markov chain approach is a good choice for counting

paths. For a complete overview of known results counting for self avoiding walks [18] provides a good reference.

Finally, Sinclair has extended the approach to sampling and counting as defined by Jerrum, Valiant, and Vazirani to weighted graphs ([24]). In order to use this approach, the bounds on the mixing time need to reflect the addition of the weights. This means that instead of merely being able to conclude the number of elements in a structure, one can now determine the sum of the weights of the elements of the structure. Therefore, in order to determine the number of paths in an arbitrary graph, one can define a weight function for the complete graph. The weight function can then be defined such that the weight of paths not in the underlying graph (called phantom paths) are negligible, and weights of paths in the underlying graph are 1. In this way, the sum of all of the weights will be very close to the number of paths in the graph. The only discrepancy will be the sum of the weights of the phantom paths.

3.3 Counting Unweighted Binary Strings

This section will detail how the Markov chain based approach is used in a simple example. To do this, we will prove that there exists an FPRAS for counting the number of binary strings of length k.

The major barrier for proving the existence of an FPRAS is the proof of the reduction of *approximate counting* to *almost uniform sampling*. This will then be used to motivate the search for an FPAUS for generating a random binary string of length k, as if an FPAUS exists, the addition of the reduction will mean an FPRAS also exists.

Theorem 3.3.1. *Let $\Sigma = \{0, 1\}$ be the alphabet to be considered, and $\mathcal{T}_i^k$ be the set of all strings over Σ of length k (i.e. Σ^k) with the last $k - i$ positions fixed to some $a \in \Sigma$. We then denote $\mathcal{T}_k^k$ as $\mathcal{T}^k$. If there exists an almost uniform sampler $\mathcal{S}$ for $\mathcal{T}_i^k$ with a polynomial run-time bounded by $T(k, \delta)$, then there is a randomized approximation scheme for $|\mathcal{T}^k|$ which runs in fully polynomial time. In particular, if there exists an FPAUS for $\mathcal{T}_i^k$, then*

there exists an FPRAS for $|\mathcal{T}^k|$ which runs in time: $75k^2\epsilon^{-2}T(k, \epsilon/6k)$.

Proof of Theorem 3.3.1. Let $\mathcal{S}$ be the almost uniform sampler from the theorem, and Σ be the alphabet to be considered. Given i, $1 \leq i \leq k$, $\mathcal{T}_{i-1}^k$ results from setting the i^{th} element of all of the strings in $\mathcal{T}_i^k$ to be a fixed element $a \in \Sigma$. We can then approximate $|\mathcal{T}^k|$ as a product of ratios[1]:

$$|\mathcal{T}^k| = (\varrho_1 \varrho_2 \cdots \varrho_k)^{-1}$$

where:

$$\varrho_i = \frac{|\mathcal{T}_{i-1}^k|}{|\mathcal{T}_i^k|}$$

(Note that $\mathcal{T}_0^k = \{\Lambda\}$, and thus $|\mathcal{T}_0^k| = 1$.) Since $\mathcal{T}_i^k$ contains all of the strings in $\mathcal{T}_{i-1}^k$, $\mathcal{T}_{i-1}^k \subseteq \mathcal{T}_i^k$. Also note that $\mathcal{T}_i^k$ can be mapped into $\mathcal{T}_{i-1}^k$. This can be done by fixing the last element t_i in all strings in $\mathcal{T}_i^k$ to a. Since there are 2 choices for t_i in $\mathcal{T}_i^k$ for each prefix, there will be at most 2 strings mapped to each string in $\mathcal{T}_{i-1}^k$. Hence,

$$\frac{1}{2} \leq \varrho_i \leq 1. \tag{3.1}$$

(Note that while this inequality appears trivial, similar bounds can be proven for problems with no known closed form expression.)

Since we want an $e^{\pm\epsilon}$–approximation, we may assume $0 < \epsilon \leq 1$, and $k \geq 1$ to avoid trivial strings. Then in order to estimate ϱ_i, we run the sampler $\mathcal{S}$ on $\mathcal{T}_i^k$ with some δ – which will be computed precisely later – in order to obtain a random string T_i from $\mathcal{T}_i^k$. We then define Z_i as the indicator variable of the event that T_i is in $\mathcal{T}_{i-1}^k$, and let $\mu_i = \mathcal{E}(Z_i) = \Pr[Z_i = 1]$. (Note that this is equivalent to the probability of the i^{th} position in T_i being a.) From the definitions of δ, and the variation distance,

$$\varrho_i - \delta \leq \mu_i \leq \varrho_i + \delta$$

And combining with (3.1)

$$(1 - 2\delta)\varrho_i \leq \mu_i \leq (1 + 2\delta)\varrho_i \tag{3.2}$$

[1]Informally, whenever we can express a counting problem as a product of polynomial-number of ratios, each of which is ratio of counts for subproblems, the problem is called self-reducible.

So the sample mean of a sufficiently large number s of independent copies $Z_i^{(1)}, \ldots, Z_i^{(s)}$ of the random variable Z_i will provide a good estimate for ϱ_i, assuming δ provides a good bound.

Before continuing to try to find s, it will be useful to have a lower bound on μ_i. In order to do this, we must define δ. Keeping in line with the bound in the original proof in Jerrum ([15]), we will let $\delta = \epsilon/6k$. Combining this with (3.2), we have:

$$\left(1 - \frac{\epsilon}{3k}\right) \varrho_i \leq \mu_i \leq \left(1 + \frac{\epsilon}{3k}\right) \varrho_i \tag{3.3}$$

Therefore we can get a lower bound for μ_i by noting that (3.3) is minimized when $\epsilon = 1$, $k = 1$, and $\varrho_i = \frac{1}{2}$, yielding: $\mu_i \geq 1/3$.

Next, we let $\overline{Z}_i = s^{-1} \sum_{j=1}^{s} Z_i^{(j)}$. Note that $\text{Var}[Z_i] = \mathcal{E}[(Z_i - \mu_i)^2] = \Pr[Z_i = 1](1 - \mu_i)^2 + \Pr[Z_i = 0]\mu_i^2 = \mu_i(1 - \mu_i)$. By our previously computed lower bound, we have: $\mu_i^{-2}\text{Var}[Z_i] = \mu_i^{-1} - 1 \leq 3/1 - 1 \leq 2$, and therefore:

$$\frac{\text{Var}(\overline{Z}_i)}{\mu_i^2} \leq \frac{2}{s} = x \tag{3.4}$$

In order to estimate $|\mathcal{T}_k|$, we will use the random variable:

$$\mathcal{N} = \prod_{i=1}^{k} \overline{Z}_i.$$

Note: $\mathcal{E}[\overline{Z}_1 \overline{Z}_2 \cdots \overline{Z}_k] = \mu_1 \mu_2 \cdots \mu_k$, and

$$\begin{aligned}
\frac{\text{Var}[\overline{Z}_1 \overline{Z}_2 \cdots \overline{Z}_k]}{(\mu_1 \mu_2 \cdots \mu_k)^2} &= \frac{\mathcal{E}[\overline{Z}_1^2 \overline{Z}_2^2 \cdots \overline{Z}_k^2]}{(\mu_1 \mu_2 \cdots \mu_k)^2} - 1 \\
&= \prod_{i=1}^{k} \frac{\mathcal{E}[\overline{Z}_i^2]}{\mu_i^2} - 1 && \text{(since the } \overline{Z}_i\text{'s are independent)} \\
&= \prod_{i=1}^{k} \left(1 + \frac{\text{Var}[\overline{Z}_i]}{\mu_i^2}\right) - 1 \\
&\leq (1 + x)^k - 1 && \text{(by (3.4))} \\
&\leq e^y - 1 && \text{(where } x = y/k\text{)} \\
&\leq \frac{z}{c} && \text{(where } y = z/(c+1)\text{, and } c \in \mathbb{N}^+\text{)}
\end{aligned}$$

since $e^{a/(b+1)} \leq 1 + a/b$ for $0 \leq a \leq 1$ and $b \in \mathbb{N}^+$. (Note that s must conform to all of these intermediate constraints.)

The final major task is to bound the probability that $\overline{Z}_1\overline{Z}_2 \cdots \overline{Z}_k$ is close to $\varrho_1\varrho_2 \cdots \varrho_k$. Chebyshev's Inequality states:

$$\Pr(|X - \mu| \geq b) \leq \frac{\text{Var}[X]}{b^2}$$

Therefore, by Chebyshev's Inequality, and by letting $b = \frac{\epsilon\mu_1\mu_2\cdots\mu_k}{3}$, we have the bound:

$$\left(1 - \frac{\epsilon}{3}\right)\mu_1\mu_2 \cdots \mu_k \leq \overline{Z}_1\overline{Z}_2 \cdots \overline{Z}_k \leq \left(1 + \frac{\epsilon}{3}\right)\mu_1\mu_2 \cdots \mu_k \tag{3.5}$$

with probability at least (letting $a = z/c$) $1 - (\epsilon/3)^{-2}a$. Since we want the error rate of this algorithm to be $1/4$, the probability needs to be $3/4$. Solving for a:

$$\begin{aligned} 1 - \left(\frac{\epsilon}{3}\right)^{-2} a &= \frac{3}{4} \\ \left(\frac{\epsilon}{3}\right)^{-2} a &= \frac{1}{4} \\ a &= \left(\frac{\epsilon}{3}\right)^{2} \frac{1}{4} \\ a &= \frac{\epsilon^2}{36} \end{aligned} \tag{3.6}$$

Therefore, now that we have a, we can solve for s. From (3.6) we can set $z = \epsilon^2$, and $c = 36$. Therefore, $y = \epsilon^2/37$, and $x = \epsilon^2/(37k)$. Thus:

$$\begin{aligned} \frac{2}{s} &= \frac{\epsilon^2}{37k} \\ s &= \left\lceil \frac{74k}{\epsilon^2} \right\rceil \\ s &\leq 75k\epsilon^{-2} \end{aligned}$$

We now have the s required by the algorithm.

Returning to (3.5), we see that we can weaken the inequality to be:

$$\left(e^{-\epsilon/2}\right)\mu_1\mu_2 \cdots \mu_n \leq \overline{Z}_1\overline{Z}_2 \cdots \overline{Z}_n \leq \left(e^{-\epsilon/2}\right)\mu_1\mu_2 \cdots \mu_n. \tag{3.7}$$

The same weakening technique can be used on (3.3) to obtain:

$$\left(e^{-\epsilon/2}\right) \varrho_1 \varrho_2 \cdots \varrho_n \leq \mu_1 \mu_2 \cdots \mu_n \leq \left(e^{-\epsilon/2}\right) \varrho_1 \varrho_2 \cdots \varrho_n \tag{3.8}$$

Combining (3.7) and (3.8), we see we have:

$$\left(e^{-\epsilon}\right) \varrho_1 \varrho_2 \cdots \varrho_n \leq \overline{Z}_1 \overline{Z}_2 \cdots \overline{Z}_n \leq \left(e^{-\epsilon}\right) \varrho_1 \varrho_2 \cdots \varrho_n \tag{3.9}$$

with probability at least $3/4$.

We now note that $\overline{Z}_1 \overline{Z}_2 \cdots \overline{Z}_n = \mathcal{N}^{-1}$, and $\varrho_1 \varrho_2 \cdots \varrho_n = |\mathcal{T}_k|^{-1}$. This means that the previously defined estimator $\mathcal{N}$ satisfies (2.1). Therefore, the algorithm computing $\mathcal{N}$ is an FPRAS for $|\mathcal{T}_k|$, as desired.

We finally note the running time of this algorithm. Since for $0 < i \leq k$ we must compute s samples, we have that the algorithm is at least $\mathcal{O}(75k^2\epsilon^{-2}) = \mathcal{O}(k^2\epsilon^{-2})$. We also must take into account the time per sample, which is $T(k, \epsilon/6k)$ (as the sampler rate requires an error rate of $\epsilon/6k$ from the sampler), and therefore the algorithm will run in time $75k^2\epsilon^{-2}T(k, \epsilon/6k)$, and the claimed bound has been proven. □

In this proof we have shown how to develop an $e^{\pm\epsilon}$ algorithm for the cardinality of a set. In order to do this, we first noted that the expected value of our defined indicator variable was in fact the ratio of the cardinalities of the set and its subproblem. In order to get a good estimate of the expected value of the indicator variable, we ran the FPAUS for $75k\epsilon^{-2}$ iterations, taking the expected value of all of the indicator variables we accumulated. Once we obtain the expected value of the indicator variable for all k subproblems, we multiply the ratios to obtain an estimate on the cardinality of the set. We then note that by our definition this estimate will be accurate with probability $3/4$, and to obtain a more accurate prediction we can use the boosting algorithm, Algorithm (1).

In this way we have shown that an FPAUS for binary strings will lead to an FPRAS for counting the number of such strings. We therefore now show how to define such a sampler as required by the algorithm.

Theorem 3.3.2. *There exists an FPAUS for unweighted binary strings with run-time bounded by:* $4k^2\left(2\ln(\delta) + k\ln(2)\right)$.

Proof of Theorem 3.3.2. Once again let Σ be the alphabet of characters we are considering, where $|\Sigma| = 2$. Let $M_\Sigma = \{N_t : t \in \mathbb{N}_0\}$ be a Markov chain such that: N_t is a binary string of length k in Σ^k, and M_Σ has state space Ω. We then define the transitions from each state $x = x_1 \dots x_k \in \Omega$ as follows:

UNWEIGHTED-BINARY-STRING-TRANSITION(x)
1 With probability $\frac{1}{2}$ let $y = x$; otherwise
2 Select i u.a.r.[a] from the range $1 \leq i \leq k$
3 Let $y = x$ with the ith bit flipped.
4 **return** y

[a]Note we use u.a.r. for *uniformly at random*

Algorithm 2: *Markov chain which defines how to transition between unweighted binary strings.*

Informally, the process M_Σ moves around the binary strings of length k by flipping a random bit at every step.

The Markov chain M_Σ is ergodic since all states communicate (to see this, note that from any state one can get to the state of all 0s by flipping all of the 1s in the string to 0s, and conversely from all 0s one can get to any state by flipping the appropriate 0s to 1s), and line (1) ensures aperiodicity. Also note that the probability of transitioning from state x to y is the same as transitioning from y to x, and thus the Markov chain is symmetric. Therefore the stationary distribution over Ω is unique and uniform.

We can next define a canonical path between two states $x = x_1 \dots x_k$ and $y = y_1 \dots y_k$ (where $x, y \in \Omega$). The canonical path γ_{xy} from x to y is defined by (at most) k edges e_1 to e_k, where edge e_i is defined as: $(y_1 \dots y_{i-1}x_i \dots x_k, y_1 \dots y_{i-1}y_i x_{i+1} \dots x_k)$, i.e. we flip the ith bit in x to be the same as y_i. In some instances no change is necessary (since $x_i = y_i$ by chance), and therefore the edge will be a self loop. In order to compute $\overline{\rho}$, we consider a

particular (oriented) edge:

$$e = (p, p') = \left(p'_1 \dots p'_{i-1} p_i \dots p_k, p'_1 \dots p'_i p_{i+1} \dots p_k\right)$$

Define $cp(e) = \{(x, y) : \gamma_{xy} \ni e\}$ the set of all canonical paths which go through e. We can then define the injective function $\eta_e : cp(e) \to \Omega$ as follows: if $(x, y) = (x_1 \dots x_k, y_1 \dots y_k) \in cp(e)$ then:

$$\eta_e(x, y) = (x_1 \dots x_{i-1} y_i \dots y_k)$$

That is, $\eta_e(x, y)$ (called an encoding) first encodes the original values of the bits which have already been set (i.e. $x_1 \dots x_{i-1}$), and then encodes the values of the bits which have not yet been reached (i.e. $y_i \dots y_k$).

For example, let $k = 7$, the initial string be $x = 0100010$, and the final string be $y = 1011001$. Also let the transition $e(x, y) = \{1010010, 1011010\}$. From this we can infer $i = 4$, and therefore, by definition, $\eta_e = 0101001$.

Note that η_e is injective since we can recover x and y unambiguously from e and $(u_1 \dots u_k) = \eta_e(x, y)$ (where e is defined as above). In order to recover x, we note that $x_1 \dots x_{i-1}$ are immediately recoverable by the definition of $\eta_e(x, y)$. In order to recover $x_i \dots x_k$, we note that $e = (p, p')$, where $p = y_1 \dots y_{i-1} x_i \dots x_k$. Therefore we can recover $x_i \dots x_k$ from p. In our example, $x_1 x_2 x_3 = u_1 u_2 u_3 = 010$, and $x_4 x_5 x_6 x_7 = p_4 p_5 p_6 p_7 = 0010$. Therefore, $x = 0100010$, and thus x was recovered correctly.

To recover y, we again note that $e = (p, p')$, where $p = y_1 \dots y_{i-1} x_i \dots x_k$. Therefore, from p we can recover $y_1 \dots y_{i-1}$. To recover the rest of y, we note that $y_i \dots y_k$ are recoverable from $u = \eta_e(x, y) = x_1 \dots x_{i-1} y_i \dots y_k$. Therefore, $y_i \dots y_k = u_i \dots u_k$. In our example, $y_1 y_2 y_3 = p_1 p_2 p_3 = 101$, and $y_4 y_5 y_6 y_7 = u_4 u_5 u_6 u_7 = 1001$. Thus $y = 1011001$, and was thus y was recovered correctly.

Using the fact that η_e is injective, we can now evaluate $\overline{\rho}$. Note that: $\pi(x)\pi(y) =$

$\pi(p)\pi(\eta_e(x,y))$ (since the stationary distribution is uniform), and thus:

$$\begin{aligned}\frac{1}{\pi(p)P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(p)\pi(\eta_e(x,y))|\gamma_{xy}| &\leq \frac{k}{P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(\eta_e(x,y)) \\ &\leq \frac{k}{P(p,p')} \\ &\leq \frac{k}{\frac{1}{2k}} \\ &\leq 2k^2\end{aligned}$$

We have that $|\gamma_{xy}| \leq k < k$ by definition, and $\sum_{\gamma_{xy}\ni e}\pi(\eta_e(x,y)) \leq 1$ since π is a probability distribution (and thus the sum over all elements is 1), and $\eta_e(x,y)$ is injective (and thus relates to, at most, all of the elements in π). Therefore, we have that $\overline{\rho} < 2k^2$. The claimed bound follows immediately once we note that there are at most 2^k binary strings, and thus $\pi(x) > 1/2^k$. □

With these two proofs we can now prove the existence of an FPRAS for approximating the size a combinatorial structure which is (potentially) exponential. To do this we note the following theorem.

Theorem 3.3.3. *There exists an FPRAS for counting the number of binary strings of length k which runs in time:* $300k^4\epsilon^{-2}\left(2\ln(6k/\epsilon) + k\ln(2)\right)$.

Proof of Theorem 3.3.3. By combining Theorem 3.3.1 with Theorem 3.3.2 (and noting the requirement for δ in Theorem 3.3.1) the claimed bound is immediate. □

While this proof only holds for unweighted structures, in the next section we discuss the weighted case. This eases the restriction and makes the technique even more general.

3.4 Counting Weighted Binary Strings

This section will proceed analogously to the unweighted section, focusing on the main results for the weighted case. Since the results are in some cases so similar, these proofs will refer to their unweighted counterparts when appropriate.

Since this section deals with weighted strings, we will define weight as follows[2]. For each position $1 \leq i \leq k$ in a string, define $c(a_i)$ to be the weight of character $a \in \Sigma$ in position i. If $x = x_1 \dots x_k$ is a string of length k, the weight of x is defined as: $w(x) = \prod_{i=1}^{k} c(x_i)$. Finally we define $w_{\min}$ to be the minimum weight of any character, and $w_{\max}$ to be the maximum weight of any character.

Since we are dealing with weighted strings, we note that the distribution from which we are sampling is a weighted distribution. That is, the probability of a string occurring is proportional to its weight, as opposed to the unweighted case where it merely depended upon the total number of strings. The weighted count, therefore, will determine the sum of all of these weights, as opposed to the cardinality of the set, similarly to the partition function defined previously.

Once again we will motivate the search for an FPAUS which generates a weighted binary string of length k by showing that if an FPAUS exists, then an FPRAS also exists.

Theorem 3.4.1. *Let $\Sigma = \{0, 1\}$ be an alphabet, and $\mathcal{T}_i^k$ be the set of all weighted strings of length k (i.e. Σ^k) with the last $k - i$ positions fixed to some $a_{k-i} \in \Sigma$. If there is an almost uniform sampler for $\mathcal{T}_i^k$ with a run-time bounded by $T(k, \delta)$, then there is a randomized approximation scheme for $w(\mathcal{T}^k)$ which runs in fully polynomial time. In particular, if there exists an FPAUS for $\mathcal{T}_i^k$, then there exists an FPRAS for $w(\mathcal{T}^k)$ which runs in time $75k^2\epsilon^{-2}T(k, \epsilon/6k)$.*

Proof of Theorem 3.4.1. This proof proceeds exactly as the unweighted case, with some minor alterations.

As before we approximate $w(\mathcal{T}_k)$ as a product of ratios:

$$w(\mathcal{T}^k) = (\varrho_1 \varrho_2 \cdots \varrho_k)^{-1}$$

where:

$$\varrho_i = \frac{w(\mathcal{T}_{i-1}^k)}{w(\mathcal{T}_i^k)}$$

[2]Note that this is not the only way to define weights, but is particularly common in physics.

(Note again that $\mathcal{T}_0^k = \{\Lambda\}$, and thus $w(\mathcal{T}_0^k) = 1$ as it is the empty product.) Since $\mathcal{T}_i^k$ contains all of the strings in $\mathcal{T}_{i-1}^k$, $\mathcal{T}_{i-1}^k \subseteq \mathcal{T}_i^k$. As before, we must determine a lower bound for ϱ_i. Since we have fixed the ith character in $\mathcal{T}_{i-1}^k$ to be a_i (which is either 0 or 1 and denoted $\mathcal{T}_{i,a}^k$), we have the following:

$$\varrho_i = \frac{w(\mathcal{T}_{i,a}^k)}{w(\mathcal{T}_{i,0}^k) + w(\mathcal{T}_{i,1}^k)}$$

In order to determine the ratio, we must formally define a_i. Note that if a_i is defined to be the character with the higher weight (*i.e.* $c(a_i) \geq c(j_i)$ for all $j \in \Sigma$), we note that $\varrho_i \geq 1/2$ (since it contains at least half of the probability). Combining the upper and lower bounds, we have:

$$\frac{1}{2} \leq \varrho_i \leq 1. \tag{3.10}$$

Since the bound on ϱ_i is the same, the rest of the proof will proceed exactly as before. This is a curiosity due to the fact that we can bound ϱ_i by a constant, as opposed to it depending on the weight function. In other problems it is possible for ϱ_i to depend on the weight. In these instances the running time would then be influenced by the weight similarly as the sampling algorithm defined below.

Therefore, as before, the algorithm will run in time $75k^2\epsilon^{-2}T(k, \epsilon/6k)$. □

Theorem 3.4.2. *There exists an FPAUS for weighted binary strings with run-time bounded by:* $4k^2 w_{\max}/w_{\min}\,(2\ln(\delta) + k\ln(2w_{\max}/w_{\min}))$*, where* $w_{\min}$ *is the minimum weight in* $c(\cdot)$*, and* $w_{\max}$ *is the maximum weight in* $c(\cdot)$*.*

Proof of Theorem 3.4.2. Using the definitions of Σ, $c(\cdot)$, $w_{\min}$, and $w_{\max}$, define $M_\Sigma = \{N_t : t \in \mathbb{N}_0\}$ to be a Markov chain such that: N_t is a binary string of length k in Σ^k, and M_Σ has state space Ω. We then define $\pi(x)$ as follows:

$$\pi(x) = \frac{w(x)}{w(\Omega)}$$

where $w(\Omega)$ is the quantity to be determined. While this definition seems strange, we will show how the dependence on $w(\Omega)$ does not matter.

WEIGHTED-BINARY-STRING-TRANSITION(x)

1 With probability $\frac{1}{2}$ let $y = x$; otherwise
2 Select i u.a.r. from the range $1 \leq i \leq k$
3 Let $z = x$ with the ith bit flipped.
4 With probability $\min\{1, \pi(z)/\pi(x)\}$ let $y = z$; otherwise
5 Let $y = x$.
6 **return** y

Algorithm 3: *Markov chain which defines how to transition between weighted binary strings.*

We then define the transitions from each state $x = x_1 \dots x_k \in \Omega$ as follows: Informally, the process M_Σ moves around the binary strings of length k. As opposed to moving around completely randomly as in the unweighted case, step (4) allows the Markov chain to prefer to transition to strings with a higher weight by using the Metropolis-Hastings algorithm.

As before, the Markov chain M_Σ is ergodic since all states communicate, and the self loops ensure aperiodicity. Since the Markov chain is ergodic, it has a unique stationary distribution over Ω. Note that as we defined the Markov chain, $\pi(\cdot)$ satisfies the detailed balance condition, and therefore is the unique stationary distribution. Also note that $\pi(z)/\pi(x)$ is easy to compute, as the $w(\Omega)$ cancel leaving $w(z)/w(x)$. From the definition of $\pi(\cdot)$ we see that the stationary distribution is proportional to the weights, denoted: $\pi(x) \propto w(x)$.

The canonical path between two states will be defined exactly as before. Thus γ_{xy} is defined by (at most) k edges, e_1 to e_k, where edge e_i is defined as:
$(y_1 \dots y_{i-1} x_i \dots x_k, y_1 \dots y_{i-1} y_i x_{i+1} \dots x_k)$, i.e. we flip the ith bit in x to be the same as y_i. In order to compute $\overline{\rho}$, we consider a particular (oriented) edge:

$$e = (p, p') = \left(p'_1 \dots p'_{i-1} p_i \dots p_k, p'_1 \dots p'_i p_{i+1} \dots p_k\right)$$

Once again, define $cp(e) = \{(x, y) : \gamma_{xy} \ni e\}$ as the set of all canonical paths which go through e. We can then define the injective function $\eta_e : cp(e) \to \Omega$ as follows: if $(x, y) = (x_1 \dots x_k, y_1 \dots y_k) \in cp(e)$ then:

$$\eta_e(x, y) = (x_1 \dots x_{i-1} y_i \dots y_k)$$

Note that this is exactly as we defined the canonical path and the encoding in the unweighted case, so the same properties still hold (namely η_e is injective).

The difference between the weighted case, and the unweighted case, is the importance of $w(p)w(\eta_e(x,y)) \approx w(x)w(y)$. This in turn will help bound the quantity of interest, namely: $\pi(p)\pi(\eta_e(x,y)) \approx \pi(x)\pi(y)$. In this example, the equality $w(p)w(\eta_e(x,y)) = w(x)w(y)$ holds as:

$$\begin{aligned}
w(p)w(\eta_e(x,y)) &= \prod_{j=1}^{k} w(p_j) \prod_{m=1}^{k} w(\eta_e(x,y)) \\
&= \left(\prod_{j=1}^{i-1} w(y_j) \prod_{j'=i}^{k} w(x_{j'})\right)\left(\prod_{m=1}^{i-1} w(x_m) \prod_{m'=i}^{k} w(y_{m'})\right) \\
&= \prod_{\overline{j}=1}^{k} w(x_{\overline{j}}) \prod_{\overline{m}=1}^{k} w(y_{\overline{m}}) \\
&= w(x)w(y)
\end{aligned}$$

Since $w(p)w(\eta_e(x,y)) = w(x)w(y)$, and $\pi(\cdot) \propto w(\cdot)$, we have that $\pi(p)\pi(\eta_e(x,y)) = \pi(x)\pi(y)$. Thus as before we see that (since η_e is still injective):

$$\begin{aligned}
\frac{1}{\pi(p)P(p,p')} \sum_{\gamma_{xy} \ni e} \pi(p)\pi(\eta_e(x,y))|\gamma_{xy}| &\leq \frac{k}{P(p,p')} \sum_{\gamma_{xy} \ni e} \pi(\eta_e(x,y)) \\
&\leq \frac{k}{P(p,p')} \\
&\leq \frac{k}{w_{\min}/2kw_{\max}} \\
&\leq \frac{2k^2 w_{\max}}{w_{\min}}
\end{aligned}$$

Note once again that $|\gamma_{xy}| \leq k$ by definition, and $\sum_{\gamma_{xy} \ni e} \pi(\eta_e(x,y)) \leq 1$ since π is a probability distribution and $\eta_e(x,y)$ is injective. Also, we note that the smallest $P(p,p')$ can be corresponds flipping a bit with a weight that is most likely ($w_{\max}$) to the least likely ($w_{\min}$) Therefore, $\overline{\rho} \leq 2k^2 w_{\max}/w_{\min}$.

Finally we bound $\pi(x)$. To do this, we first note that $w(\Omega)$ is bounded above by $2^k w_{\max}^k$. Therefore, noting that the minimum weight of a string is $w_{\min}^k$, $\pi \geq w_{\min}^k/(2w_{\max})^k$, and the bound is immediate. $\square$

Given this, we can now prove the following.

Theorem 3.4.3. *There exists an FPRAS for computing the sum of the weights of all binary strings of length k with run time:* $300k^4\epsilon^{-2}w_{\max}/w_{\min}\left(2\ln(6k/\epsilon) + k\ln(2w_{\max}/w_{\min})\right)$.

Proof of Theorem 3.4.3. By combining Theorem 3.4.1 with Theorem 3.4.2 (and noting the requirement for δ in Theorem 3.4.1) the claimed bound is immediate. □

Note that for this specific problem, the terms $w_{\max}/w_{\min}$ in the running time can be avoided, yielding a truly polynomial-time bound. Since the required techniques do not appear to apply to most other weighted problems, and this is just an example to demonstrate the technique, we opted not to discuss them here.

In the weighted case we have shown how to estimate the size of a structure when each element is not counted equally. The only restriction that remains is the dependence on the weights. More specifically, the running time of the algorithm depends on the weights, and thus for the algorithm to run in polynomial time, the weights must be polynomial in the input size. If one is willing to run in exponential time, then the weights can be exponential, however since the structure to be estimated has (potentially) exponentially many values, in that case the approximation algorithm may not be needed.

Chapter 4

Counting From Sampling

In this chapter we will outline algorithms to sample almost *uniformly at random* (u.a.r.) from various combinatorial structures. The specific structures to be consider are the weighted and unweighted complete graph, and the weighted and unweighted tree. When an FPRAS exists (*e.g.* when the Markov chain is rapidly mixing), the proofs will proceed in the same way as the binary string example presented as an example. For the negative examples (i.e. when the Markov chain is torpidly mixing), the proof of torpid mixing will be presented.

4.1 Unweighted Complete Graph

In this section we will outline the FPRAS for the unweighted complete graph. To do so, we will begin with the reduction from approximate counting to almost uniform sampling. The reduction will then motivate the search for an FPAUS, which will then be presented.

4.1.1 *Approximate Counting* $\leq$ *Almost Uniform Sampling*

Theorem 4.1.1. *Given a graph* $G = (V, E)$ *where* $|V| = n$*, and* $|E| = m$*, and a FPAUS with running time bounded by* $T(n, m, \delta)$*, there exists a FPRAS for counting the number of paths in* G $(|\mathcal{P}(G)|)$ *which runs in time* $56(n+1)n^2\epsilon^{-2}T(n, m, \epsilon/3n(n+1))$.

Proof of Theorem 4.1.1. Let $\mathcal{S}$ be the almost uniform sampler from the theorem, then the approximation scheme proceeds as follows. Let $G = (V, E)$ be as defined in the theorem

such that $V = \{v_1, v_2, \ldots, v_n\}$, then G_i is the vertex induced subgraph resulting from the first i vertices $(v_0, v_1, \ldots v_i)$ in G, $0 < i \leq n$. Therefore, given G_i, G_{i-1} results from the deletion of the vertex v_i. We can then approximate $|\mathcal{P}(G)|$ as a product of ratios:

$$|\mathcal{P}(G)| = (\varrho_1 \varrho_2 \cdots \varrho_m)^{-1}$$

where:

$$\varrho_i = \frac{|\mathcal{P}(G_{i-1})|}{|\mathcal{P}(G_i)|}$$

(Note that $|\mathcal{P}(G_0)| = 1$ since it contains the trivial path of no vertices.) Since G_i contains all of the paths in G_{i-1}, $\mathcal{P}(G_{i-1}) \subseteq \mathcal{P}(G_i)$. Also, $\mathcal{P}(G_i) \setminus \mathcal{P}(G_{i-1})$ can be mapped into $\mathcal{P}(G_{i-1})$. This can be done by simply removing vertex $\{v_i\}$ from the paths in $\mathcal{P}(G_i) \setminus \mathcal{P}(G_{i-1})$ (i.e. send all of the paths to $\mathcal{P} \setminus \{v_i\}$). Since there are at most i positions for vertex $\{v_i\}$ to appear in a path of length i, there will be at most $i + 1$ paths mapped to each $\mathcal{P} \setminus \{v_i\}$. Hence,

$$\frac{1}{i+1} \leq \varrho_i \leq 1. \tag{4.1}$$

Since we want an $e^{\pm\epsilon}$–approximation, we may assume $0 < \epsilon \leq 1$, and $n \geq 1$ to avoid trivial graphs. Then in order to estimate ϱ_i, we run the sampler $\mathcal{S}$ on G_i with some δ – which will be computed precisely later – in order to get a random path P_i from $\mathcal{P}(G_i)$. We then define Z_i as the indicator variable of the event that P_i is in $\mathcal{P}(G_{i-1})$, and let $\mu_i = \mathcal{E}(Z_i) = \Pr[Z_i = 1]$. From the definitions of δ, and the variation distance,

$$\varrho_i - \delta \leq \mu_i \leq \varrho_i + \delta$$

And combining with (4.1)

$$(1 - (i+1)\delta)\varrho_i \leq \mu_i \leq (1 + (i+1)\delta)\varrho_i. \tag{4.2}$$

Thus the sample mean of a sufficiently large number s of independent copies $Z_i^{(1)}, \ldots, Z_i^{(s)}$ of the random variable Z_i will provide a good estimate for ϱ_i, assuming δ provides a good bound.

Before continuing trying to find s, it will be useful to have a lower bound on μ_i. Letting $\delta = \frac{\epsilon}{3n(i+1)}$ and combining with (4.2), we have:

$$\begin{aligned}\left(1 - \frac{\epsilon(i+1)}{3n(i+1)}\right)\varrho_i \leq &\mu_i \leq \left(1 + \frac{\epsilon(i+1)}{3n(i+1)}\right)\varrho_i \\ \left(1 - \frac{\epsilon}{3n}\right)\varrho_i \leq &\mu_i \leq \left(1 + \frac{\epsilon}{3n}\right)\varrho_i \end{aligned} \tag{4.3}$$

Therefore we can get a lower bound for μ_i by noting that (4.3) is minimized when $\epsilon = 1$, $n = 1$, and $\varrho_i = \frac{1}{i+1}$, yielding: $\mu_i \geq 2/(3(i+1))$.

Next, we let $\overline{Z}_i = s^{-1}\sum_{j=1}^{s} Z_i^{(j)}$. Note that $\text{Var}[Z_i] = \mathcal{E}[(Z_i - \mu_i)^2] = \Pr[Z_i = 1](1-\mu_i)^2 + \Pr[Z_i = 0]\mu_i^2 = \mu_i(1-\mu_i)$. By our previously computed lower bound, we have: $\mu_i^{-2}\text{Var}[Z_i] = \mu_i^{-1} - 1 \leq \frac{3(i+1)}{2} - 1 < \frac{3(i+1)}{2}$, and therefore:

$$\frac{\text{Var}(\overline{Z}_i)}{\mu_i^2} < \frac{3(i+1)}{2s} = x \tag{4.4}$$

In order to estimate $|\mathcal{P}(G)|$, we will use the random variable:

$$\mathcal{N} = \prod_{i=1}^{n} \overline{Z}_i.$$

Note: $\mathcal{E}[\overline{Z}_1\overline{Z}_2\cdots\overline{Z}_n] = \mu_1\mu_2\cdots\mu_n$, and

$$\begin{aligned}\frac{\text{Var}[\overline{Z}_1\overline{Z}_2\cdots\overline{Z}_n]}{(\mu_1\mu_2\cdots\mu_n)^2} &= \frac{\mathcal{E}[\overline{Z}_1^2\overline{Z}_2^2\cdots\overline{Z}_n^2]}{(\mu_1\mu_2\cdots\mu_n)^2} - 1 \\ &= \prod_{i=1}^{n}\frac{\mathcal{E}[\overline{Z}_i^2]}{\mu_i^2} - 1 && \text{(since the } \overline{Z}_i\text{'s are independent)} \\ &= \prod_{i=1}^{n}\left(1 + \frac{\text{Var}[\overline{Z}_i]}{\mu_i^2}\right) - 1 \\ &< (1+x)^n - 1 && \text{(by (4.4))} \\ &< e^y - 1 && \text{(where } x = y/n\text{)} \\ &< \frac{z}{c} && \text{(where } y = z/(c+1)\text{, and } c \in \mathbb{N}^+\text{)}\end{aligned}$$

since $e^{x/(k+1)} \leq 1 + x/k$ for $0 \leq x \leq 1$ and $k \in \mathbb{N}^+$. (Note that s must conform to all of these intermediate constraints.)

The final major task is to bound $\overline{Z}_1\overline{Z}_2\cdots\overline{Z}_n$ by $\varrho_1\varrho_2\cdots\varrho_n$. By Chebyshev's Inequality, and by letting $k = \frac{\epsilon\mu_1\mu_2\cdots\mu_n}{3}$, we have the bound:

$$\left(1-\frac{\epsilon}{3}\right)\mu_1\mu_2\cdots\mu_n \leq \overline{Z}_1\overline{Z}_2\cdots\overline{Z}_n \leq \left(1+\frac{\epsilon}{3}\right)\mu_1\mu_2\cdots\mu_n \tag{4.5}$$

with probability at least (letting $a = z/c$) $1-(\epsilon/3)^{-2}a$. If we want this probability to be $3/4$, we solve for a as follows:

$$\begin{aligned} 1-\left(\frac{\epsilon}{3}\right)^{-2}a &= \frac{3}{4} \\ \left(\frac{\epsilon}{3}\right)^{-2}a &= \frac{1}{4} \\ a &= \left(\frac{\epsilon}{3}\right)^{2}\frac{1}{4} \\ a &= \frac{\epsilon^2}{36} \end{aligned} \tag{4.6}$$

Therefore, now that we have a, we can solve for s. We now know that $z = \epsilon^2$, and $c = 36$. Therefore, $y = \epsilon^2/37$, and $x = \epsilon^2/(37n)$. Thus:

$$\begin{aligned} \frac{3(i+1)}{2s} &= \frac{\epsilon^2}{37n} \\ s &= \left\lceil \frac{111(i+1)n}{2\epsilon^2} \right\rceil \\ s &\leq 56(i+1)n\epsilon^{-2} \\ s &\leq 56(n+1)n\epsilon^{-2} \end{aligned}$$

We now have the s required by the algorithm.

Returning to (4.5), we see that we can weaken the inequality to be:

$$\left(e^{-\epsilon/2}\right)\mu_1\mu_2\cdots\mu_n \leq \overline{Z}_1\overline{Z}_2\cdots\overline{Z}_n \leq \left(e^{-\epsilon/2}\right)\mu_1\mu_2\cdots\mu_n. \tag{4.7}$$

The same weakening technique can be used on (4.3) to obtain:

$$(e^{-\epsilon/2})\varrho_1\varrho_2\cdots\varrho_n \leq \mu_1\mu_2\cdots\mu_n \leq (e^{-\epsilon/2})\varrho_1\varrho_2\cdots\varrho_n \tag{4.8}$$

Combining (4.7) and (4.8), we see we have:

$$(e^{-\epsilon})\varrho_1\varrho_2\cdots\varrho_n \leq \overline{Z}_1\overline{Z}_2\cdots\overline{Z}_n \leq (e^{-\epsilon})\varrho_1\varrho_2\cdots\varrho_n \tag{4.9}$$

with probability at least $3/4$.

We now note that $\overline{Z}_1\overline{Z}_2\cdots\overline{Z}_n = \mathcal{N}^{-1}$, and $\varrho_1\varrho_2\cdots\varrho_n = |\mathcal{P}(G)|^{-1}$. This means that the previously defined estimator $\mathcal{N}$ satisfies (2.1). Therefore, the algorithm computing $\mathcal{N}$ is an FPRAS for $|\mathcal{P}(G)|$, as desired.

We finally note the running time of this algorithm. Since for $0 < i \leq n$ we must compute s samples, we have that the algorithm requires a total of $\mathcal{O}(56(n+1)n^2\epsilon^{-2}) = \mathcal{O}(n^3\epsilon^{-2})$ samples. Since each sample takes time $T(n, m, \epsilon/3n(n+1))$, the claimed bound follows immediately. □

4.1.2 Existence of an FPAUS

Given that we have an FPRAS which runs in polynomial time if we have an FPAUS for the unweighted complete graph, we note the following theorem.

Theorem 4.1.2. *There exists an FPAUS for sampling paths from the unweighted complete graph $G = (V, E)$ which runs in time $192n^3(2\ln(\delta) + n\ln(4n))$.*

Proof of Theorem 4.1.2. Let $M_{K_n} = \{N_t : t \in \mathbb{N}_0\}$ be a Markov chain such that N_i is a path of length at most n in the complete graph on n vertices (i.e. K_n). The state space Ω is thus all of the paths in K_n. We then define the transitions from each state $x = (x_1 \ldots x_n) \in \Omega$ to state y in Algorithm 4.

Informally, the process M_{K_n} is a self-avoiding random walk (with stationary moves) on the complete graph with n vertices. To accomplish this, the algorithm first chooses a vertex v. If the vertex v is in x, the algorithm chooses a position i to move v to. If v is already in the selected position, y is defined as x with v removed. Otherwise, y is the same as x with vertex v moved from its original position to position i. If v is not in x, a position i is chosen. Given i, y is x with v inserted at position i.

The Markov chain M_{K_n} is ergodic since all states communicate, and the loops ensure aperiodicity. The states can be seen to communicate since (trivially) every path can be reduced to the empty path, and any path can be built up from the empty path. Since the

UNWEIGHTED-COMPLETE-GRAPH-TRANSITION(x)

With probability $\frac{1}{2}$ let $y = x$; otherwise
Select v u.a.r. from V.
if $v \in x$, select i u.a.r. from the range $0 \leq i \leq |x|$
  then if $v = x_i$
    then $y \leftarrow x \setminus \{p\}$
  elseif $v = x_k$ where $k < i$
    then $y \leftarrow (x_1 \dots x_{k-1} x_{k+1} \dots x_{i-1} x_i x_k x_{i+1} \dots x_{|x|-1} x_{|x|})$
  elseif $v = x_k$ where $k > i$
    then $y \leftarrow (x_1 \dots x_{i-1} x_k x_i x_{k+1} \dots x_{k-1} x_{k+1} \dots x_{|x|-1} x_{|x|})$
if $v \notin x$, select i u.a.r. from the range $0 \leq i \leq |x|$
  then $y \leftarrow (x_1 \dots x_{i-1} p x_i x_{i+1} \dots x_{|x|-1} x_{|x|})$
return y

Algorithm 4: *Algorithm which defines transitions between paths in the unweighted complete graph.*

probability of moving from any state x to any other state y is the same as moving from state y to state x, the Markov chain is symmetric, and therefore the stationary distribution is uniform over Ω.

We will first define canonical paths between two states corresponding to paths of length n and bound the congestion of these paths. After this, we will extend this argument to canonical paths between any pair of states. Let $x = (x_1 x_2 \dots x_n)$ and $y = (y_1 y_2 \dots y_n)$ (where $x, y \in \Omega$) be two states corresponding to paths of length n.

The canonical path γ_{xy} from x to y is then defined by (at most) $n-1$ edges (or transitions) e_1 to e_{n-1} (since if the first $n-1$ vertices are in order, the n^{th} vertex is in order by definition), where edge e_i is defined as:

$$((y_1 \dots y_{i-1} z_1 \dots z_{n-i}), (y_1 \dots y_{i-1} y_i z'_1 \dots z'_{n-i-1})),$$

where $z_1 \dots z_{n-1}$ and $z'_1 \dots z'_{n-i-1}$ represent the ordering of the remaining vertices in the path. Informally, the i^{th} edge in the canonical path is formed by moving the vertex which is supposed to occupy position i to position i. In some instances no change is necessary (since $z_1 = y_i$ by chance), and therefore the edge will be a loop. Note that all vertices after the i^{th} vertex are in the same order as in x (where we skip over vertices in x that correspond to a

y_k, for some $k \leq i$).

In order to compute $\overline{\rho}$[1], we consider a particular (oriented) edge:

$$e = (p, p') = \big((p'_1 \dots p'_{i-1} z_1 z_2 \dots z_{n-i+1}), (p'_1 \dots p'_{i-1} p'_i z'_1 \dots z'_{n-i})\big)$$

Define $cp(e) = \{(x, y) : \gamma_{xy} \ni e\}$ as the set of all canonical paths which go through edge e. We can then define the injective function $\eta_e : cp(e) \to \Omega$ as follows: if $(x, y) = ((x_1 \dots x_n), (y_1 \dots y_n)) \in cp(e)$ then:

$$\eta_e(x, y) = u$$

where, informally, u encodes the starting positions of vertices which are in their final positions (*i.e.* $y_1 \dots y_i$), and the order of the vertices which are yet to be placed (*i.e.* $y_{i+1} \dots y_n$).

Formally, we first fill i positions in $u = u_1 u_2 \dots u_n$ as follows:

$$u_j = x_j, \text{ where } x_j = y_k \text{ for some } k \leq i$$

Let J be the set of $n - i$ indices of u which are undefined. For $1 \leq w \leq n - i$, fill in the remaining positions in u as follows:

$$u_{J_w} = y_{w+i}$$

where J_w represents the w^{th} smallest element in J.

For example, let $n = 7$, the initial path be $x = (1354627)$, and the final path be $y = (6537241)$. Also let the transition $e = (p, p') = \{(6531427), (6537142)\}$. From this we can infer $i = 4$, and initialize $u = (u_1 u_2 u_3 u_4 u_5 u_6 u_7)$. Then performing the first step, we place the first i values from p' into u in the same positions they hold in x. This leaves $u = u_1 35 u_2 6 u_3 7$. We then fill in the remaining values in p into u in the order in which they occur in y. Thus $u = (2354617)$.

Note that η_e is injective since we can recover x and y unambiguously from $(u_1 \dots u_n) = \eta_e(x, y)$, and e (where e is defined as above). In order to recover y, we note that $y_1 \dots y_i$ are

[1]Note that this is not precisely $\overline{\rho}$, merely the congestion between paths of maximal length. We will use it later to precisely compute $\overline{\rho}$.

immediately recoverable from e. In order to recover $y_{i+1} \dots y_n$, we note that the injective mapping defines the order of the remaining vertices. Thus letting z be the remaining vertices after removing vertices $y_1 \dots y_i$ from $\eta_e(x, y)$. We then have that $y = (p'_1 \dots p'_i z_1 \dots z_{n-i})$.

Continuing with the above example (where $e = (p, p') = \{(6531427), (6537142)\}$ and $u = (1352647)$) we recover y as follows.

- First note that y must start as 6537, so we let $y = (6537y_1y_2y_3)$.
- Next, remove 6537 from u, leaving 241.
- Finally, append the result to $y = (6537241)$.

Note that this is exactly the y we started with.

In order to recover x we first recover the vertices $x_1 \dots x_i$ from $\eta_e(x, y)$ as defined in the encoding, filling in the missing entries with $a_1 \dots a_{n-i}$. Given this, we note that we can now preform all of the operations to transform x into p (since p only depends on the first $i - 1$ vertices). After symbolically manipulating x, we note that the a_k for $1 \leq k \leq n - i$ now correspond to some p_j for $i < j < n$. Therefore, filling in the corresponding values in p_j for the corresponding a_k in x, we can unambiguously recover x. Therefore $\eta_e(x, y)$ is an injective function.

Using our above example again, we recover x as follows.

- First note that $i = 4$ since in edge e the last element in position is the 4th one.
- Therefore, for $i > 4$, replace those values in u with place holders and store the result in $x = (a_1 35 a_2 6 a_3 7)$
- Next perform operations on x to transform into the beginning of e (i.e. p)
  - $x = (a_1 35 a_2 6 a_3 7)$
  - $x = (6 a_1 35 a_2 a_3 7)$
  - $x = (63 a_1 5 a_2 a_3 7)$
  - $x = (635 a_1 a_2 a_3 7)$
- Thus looking at p we see that $a_1 = 1$, $a_2 = 4$, and $a_3 = 2$.
- Therefore $x = 1354627$

Note that this is exactly the x we started with.

Using the fact that η_e is injective, we can now evaluate $\overline{\rho}$. Note that: $\pi(x)\pi(y) = \pi(p)\pi(\eta_e(x,y))$, we have:

$$\begin{aligned}
\frac{1}{\pi(p)P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(x)\pi(y)|\gamma_{xy}| &= \frac{1}{\pi(p)P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(p)\pi(\eta_e(x,y))|\gamma_{xy}| \\
&\leq \frac{n}{P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(\eta_e(x,y)) \\
&\leq \frac{n}{P(p,p')} \\
&\leq \frac{n}{\frac{1}{2n^2}} \\
&\leq 2n^3 \qquad (4.10)
\end{aligned}$$

It is important to note that in this case canonical paths are only defined between paths of maximal length. This is due to the fact that a simple counting argument shows that there are at most three times as many paths whose length is not maximal as maximal paths.

To see this, let $\mathcal{P}_n(G)$ denote the number of paths of length n in G, and $\mathcal{P}_{<n}(G)$ denote the number of paths with length less than n in G. We then see that $\mathcal{P}_n(G) = n!$, and $\mathcal{P}_{<n}(G) = \sum_{i=0}^{n}\binom{n}{i}i!$, and thus:

$$\begin{aligned}
\frac{\mathcal{P}_{<n}(G)}{\mathcal{P}_n(G)} &= \frac{\sum_{i=0}^{n-1}\binom{n}{i}i!}{n!} \\
&= \frac{\sum_{i=0}^{n-1}\frac{n!}{i!(n-i)!}i!}{n!} \\
&= \frac{n!\sum_{i=0}^{n-1}\frac{1}{(n-i)!}}{n!} \\
&= \sum_{i=0}^{n-1}\frac{1}{(n-i)!} \\
&= \sum_{j=1}^{n}\frac{1}{(j)!} \\
&\leq e \\
&\leq 3
\end{aligned}$$

Where the penultimate inequality holds since: $\sum_{i=0}^{\infty}(1/i!) = e$.

Therefore a mapping exists that takes at most three paths of length less than n and maps them to a single path of length n. Given this, we must now give a bound on the number of times any transition $t = (a, b)$ is used where a is a state of the Markov chain corresponding to a path of length less than n. If the bound for t is less than the bound for e which we computed above (*i.e.* (4.10)), then the congestion between these states can be ignored. This is due to the fact that the congestion $\overline{\rho}$ is defined as the maximum of congestions through individual transitions of the Markov chain.

In order to compute the congestion of t, we first define *level* i as the states of the Markov chain which correspond to a path of length i. We then note that any state in level i has $(n - i)(i + 1) = -i^2 + ni - i + n$ transitions to level $i + 1$. This is due to the fact that at level i there are $n - i$ vertices which haven't been placed, and $i + 1$ positions in the string in which to place them. We next note that the expression is minimized when $i = 0$ and $i = n - 1$, which means there are at least n transitions between level i and level $i + 1$. We will order transitions originating at state s in level i lexicographically by their opposite endpoint. Thus if s is adjacent to two states a and b, each in level $i + 1$, $a < b$ if a occurs before b lexicographically.

We now (informally) define the canonical path from a path of length i to the state which represents a path of maximal length. To do so we will continue by induction, with the base case being the path of length 0 (*i.e.* the empty path). The empty path is adjacent to n states in level 1, and to define its canonical path, we transition via the first unused transition. Note that there exists a transition which is unused by all other canonical paths, as each state s at level i has at most $i - 1$ canonical paths passing through it. To see this, given state s at level i, we first let the states whose canonical paths pass through s from the first j levels, $0 \leq j < i$, choose the first (potentially) $i - 1$ transitions. The current state s therefore transitions via the i^{th} transition. We know that the i^{th} transition exists – and more importantly is unused – since there are at least n transitions going from s to a state in level $i + 1$.

Using the fact that each edge is used only once, we see that the maximum congestion

over all edges not between maximal length paths is:

$$\begin{aligned}
\frac{1}{\pi(p)P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(x)\pi(y)|\gamma_{xy}| &= \frac{1}{\pi(p)P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(p)\pi(\eta_e(x,y))|\gamma_{xy}| \\
&= \frac{1}{P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(\eta_e(x,y))|\gamma_{xy}| \\
&= \frac{|\gamma_{xy}|}{P(p,p')}\cdot\pi(\eta_e(x,y)) \\
&\leq \frac{n}{\frac{1}{2n^2}}\frac{1}{|\Omega|} \\
&\leq \frac{n^3}{2|\Omega|}
\end{aligned}$$

Since the congestion in the Markov chain between states of length n is larger than $n^3/2|\Omega|$, this does not represent the maximum edge for which the congestion can be defined.

Previously we computed the canonical paths between states representing paths of maximal length. In doing this, we noted that the maximum length of a canonical path between two states x and y was: $|\gamma_{xy}| < n$. We now consider the effect of computing the canonical path between two arbitrary states a and b (which need not represent paths of maximal length), where a maps to the maximal length path a', and b maps to the maximal length path b'. The canonical path will first transition from a to a' in at most n steps. It continues by transitioning from a' to b' in at most n steps, and ends by transitioning from b' to b in at most n steps. Thus it is easy to see that the canonical path between a and b has length less than $3n$, and thus an extra factor of 3 must be added.

Since we have shown that we can map (at most) 3 paths of length less than n to a single path of length n, we must consider the effect this has on the congestion. Informally, we see that every path of length n "represents" at most 4 paths. Therefore, the congestion through a transition used by a canonical path of length n to another path of length n increases by at most a factor of $4 \cdot 4 = 16$. Taking into account this factor of 16, and the previously mentioned factor or 3 for increasing the upper bound on the length of a canonical path, $\overline{\rho} \leq 2n^3 \cdot 3 \cdot 16 \leq 96n^3$.

In order to bound the mixing time, we then need a bound on $\pi(x)$. Since we know that

there are at most $4n!$ paths, $\pi(x) \geq 1/4n!$. The claimed bound follows from the fact that $\ln n! \leq n \ln n$. □

4.1.3 FPRAS for the Unweighted Complete Graph

Given the previous theorems, we can now prove the following.

Theorem 4.1.3. *There exists an FPRAS for the unweighted complete graph on n vertices which runs in time:* $10752(n+1)n^4\epsilon^{-2}(2\ln(3n(n+1)/\epsilon) + n\ln(4n))$.

Proof of Theorem 4.1.3. Combining the bounds in Theorem 4.1.1 with Theorem 4.1.2, the claimed bound is immediate. □

4.2 Weighted Complete Graph

In this section we will outline the RAS for the weighted complete graph. As before, we will begin with the reduction from approximate counting to almost uniform sampling. The reduction will then motivate the search for an FPAUS, which will then be presented.

While in the unweighted case we were able to bound the running time of the FPRAS by the number of vertices and an error rate, the weighted case offers the added challenge of the weights. While in the example proofs the weights didn't always factor into the running time, intuitively the weights seem to be more of a challenge in sampling paths. This will prove to be the case, as we will see the weights being an important factor in the running time.

4.2.1 *Approximate Counting $\leq$ Almost Uniform Sampling*

Theorem 4.2.1. *Given a graph $G = (V, E)$ where $|V| = n$, and $|E| = m$, a weight function $w : E \to \mathcal{R}$, and a FPAUS with running time bounded by $T(n, m, \delta)$, there exists a FPRAS for determining the sum of the paths in G $(w(\mathcal{P}(G)))$ which runs in time* $56(n+1)n^2\epsilon^{-2}T(n, m, \epsilon/3n(n+1))$.

Proof of Theorem 4.2.1. As before, we let $\mathcal{S}$ be the almost uniform sampler from the theorem, then the approximation scheme proceeds as follows. Let $G = (V, E)$ such that $V = \{v_1, v_2, \ldots, v_n\}$, then $G_i = (V_i, E_i)$ is the vertex induced subgraph resulting from the first i vertices $\{v_1, v_2, \ldots v_i\}$ in G, $0 < i \leq n$. And once again given G_i, G_{i-1} results from the deletion of the vertex v_i. The orderings of the vertices $(v_1, \ldots, v_n)$ will be defined shortly.

In the unweighted case the choice of vertex to remove was arbitrary, as there was no difference between any of the vertices. For the weighted case, however, we define the weight of vertex v, $w(v)$, to be the product of the weights of its incident edges. In graph G_i, the vertex to be removed v_i is defined as:

$$v_i = \min_{v \in V_i} w(v)$$

We can now approximate $w(\mathcal{P}(G))$ as a product of ratios:

$$w(\mathcal{P}(G)) = (\varrho_1 \varrho_2 \cdots \varrho_m)^{-1}$$

where:

$$\varrho_i = \frac{w(\mathcal{P}(G_{i-1}))}{w(\mathcal{P}(G_i))}$$

(Note that $w(\mathcal{P}(G_0)) = 1$ since it contains the trivial path of no vertices whose weight is defined by the empty product.) Since G_i contains all of the paths in G_{i-1}, $\mathcal{P}(G_{i-1}) \subseteq \mathcal{P}(G_i)$. Note also, that if all of the edge weights of the vertex to be removed are close to 0, the ratio $w(\mathcal{P}(G_{i-1}))/w(\mathcal{P}(G_i))$ will be close to (but never greater than) 1. Therefore the lower bound will occur when all of the edge weights of the vertex to be removed are equal to w_{max}. However when each vertex has the same weight, this is equivalent to the unweighted graph. Since the lower bound is equivalent to the case of the unweighted graph, we can lower bound ϱ_i by the same quantity, namely $1/(i+1)$. Hence

$$\frac{1}{i+1} \leq \varrho_i \leq 1. \tag{4.11}$$

Since this is exactly the same bound from the unweighted case, the proof proceeds exactly as before. Therefore the algorithm will run in time $56(n+1)n^2\epsilon^{-2}T(n, m, \epsilon/3n(n+1))$. $\square$

Note that in unweighted case there are on the order of $n!$ paths in the complete graph. Therefore being able to approximate this quantity in polynomial time (assuming an FPAUS exists) means the technique has solved the path counting problem. While ideally polynomial, the presented bound on the runtime of the sampler presented in section 4.2.2 is exponential. This means that that the approximation scheme as defined might not be fully polynomial, and will be dominated by the time it takes to sample from the distribution. If a new sampling algorithm proof can be devised which is an FPAUS, then the above proof demonstrates that the path counting problem can be solved in polynomial time.

4.2.2 Existence of an FPAUS

By Theorem 4.2.1, there exists an FPRAS for the sum of the weights of all of the paths in the weighted complete graph, if there exists a sampler for them. While the following theorem does not prove the existence of an FPAUS, it does give an AUS. Based on the proof in the weighted tree case, however, it doesn't seem likely that an FPAUS exists.

Theorem 4.2.2. *There exists an AUS for sampling paths from the weighted complete graph $G = (V, E)$ with running time bounded by: $12n^3 f(w_{\min})^2 \left(\frac{w_{\max}}{w_{\min}}\right)^{3n/2} (2\ln(\epsilon) + \ln(1/g(w_{\max}, w_{\min}, n)))$, where:*

$$f(w_{\min}) = \begin{cases} n & \text{if } w_{\min} = 1 \\ \frac{w_{\min}^{n+1}-1}{w_{\min}^{n+1}-w_{\min}^{n}} & \text{otherwise} \end{cases}$$

and,

$$g(w_{\max}, w_{\min}, n) = \min\{\frac{1}{4n!w_{\max}^n}, \frac{w_{\min}^n}{4n!w_{\max}^n}\}.$$

Proof of Theorem 4.2.2. Let $M_{K_n} = \{N_t : t \in \mathbb{N}_0\}$ be a Markov chain such that N_i is a path of length at most n in the weighted complete graph on n vertices (i.e. K_n). The state space Ω is thus all of the paths in K_n. For each pair of vertices $a, b \in K_n$, a weight function $c(a, b) = c(b, a)$ is defined. Let $w_{\min}$ be the minimum weight, and $w_{\max}$ the maximum weight in $c(\cdot)$. The weight of a path is then defined as: $w(a) = \prod_{i=1}^{|a|-1} c(a_i, a_{i+1})$. Note that

the path of a single vertex and the empty path have weight 1, as they are defined by the empty product.

We then define the transitions from each state $x = (x_1 \dots x_k) \in \Omega$, $0 \leq k \leq n$ as follows:

WEIGHTED-COMPLETE-GRAPH-TRANSITION(x)

With probability $\frac{1}{2}$ let $y = x$; otherwise
Select v u.a.r. from V
if $v \in x$
  then Select i u.a.r. from the range $0 \leq i \leq |x|$
    if $v = x_i$
      then Let $z = x \setminus \{p\}$
    elseif $v = x_k$ where $k < i$
      then $z \leftarrow (x_1 \dots x_{k-1} x_{k+1} \dots x_{i-1} x_i x_k x_{i+1} \dots x_{|x|-1} x_{|x|}$
    elseif $v = x_k$ where $k > i$
      then $z \leftarrow (x_1 \dots x_{i-1} x_k x_i x_{k+1} \dots x_{k-1} x_{k+1} \dots x_{|x|-1} x_{|x|})$
elseif $p \notin x$
  then Select i u.a.r. from the range $0 \leq i \leq |x|$
    then $z \leftarrow (x_1 \dots x_{i-1} p x_i x_{i+1} \dots x_{|x|-1} x_{|x|})$
With probability $\min\{1, \pi(z)/\pi(x)\}$ let y = z; otherwise
Let $y = x$.

Algorithm 5: *Markov chain which defines how to transition between states.*

Informally, the process M_{K_n} is a self-avoiding random walk (with stationary moves) on the weighted complete graph with n vertices, giving higher preference to paths with higher weights. It achieves this by first computing a potential next path in the same way as the unweighted case, and transitioning to the new path with probability proportional to the ratio of the weight of the new path, and the weight of the current path.

It is easy to see that by letting $\pi(x) = w(x)/w(\mathcal{P}(G))$, π is the unique stationary distribution. Therefore, as in the weighted binary string example, $\pi(z)/\pi(x)$ is easy to compute as it is merely $w(z)/w(x)$.

The Markov chain M_{K_n} is ergodic since all states communicate, and the loops ensure aperiodicity. Since the Markov chain is ergodic, it has a unique stationary distribution over

Ω. As shown, the stationary distribution will not necessarily be uniform, instead depending on the weights. From the definition, however, we see that $\pi(x) \propto w(x)$.

We can then once again define a canonical path between two states $x = (x_1 x_2 \dots x_n)$ and $y = (y_1 y_2 \dots y_n)$ (where $x, y \in \Omega$) The canonical path γ_{xy} from x to y is defined by (at most) $n - 1$ edges e_0 to e_n, where edge e_i is defined as:

$$((y_1 \dots y_{i-1} z_1 \dots z_{n-i}), (y_1 \dots y_{i-1} y_i z'_1 \dots z'_{n-i-1})),$$

Note that, once again, $z_1 \dots z_{n-1}$ and $z'_1 \dots z_{n-i-1}$ define the ordering of the remaining vertices in their respective paths. Thus, informally, to define the canonical path, the vertex which is supposed to occupy position i is "pulled" to position i. In some instances no change is necessary (since $z_1 = y_i$ by chance), and therefore the edge will be a loop. In order to compute $\overline{\rho}$, we consider a particular (oriented) edge:

$$e = (p, p') = \big((p'_0, \dots, p'_{i-1}, z_1, z_2, \dots, z_{n-i+1}), (p'_0, \dots, p'_{i-1}, p'_i, z'_1 \dots, z'_{n-i})\big)$$

It is important to note at this point that in the unweighted case we could bound the number of paths of length i by a constant times the number of paths of length less than i. In this case, however, we can only make the following observation (let $\mathcal{P}_i(G)$ denote the set of paths of length i in G, and $w(\mathcal{P}_i(G))$ denote the sum of the weights of set $\mathcal{P}_i(G)$),

$$\frac{w(\mathcal{P}_i(G))}{w_{\min}} \geq w(\mathcal{P}_{i-1}(G))$$

This can be seen since it corresponds to removing a minimally weighted edge from every path in $\mathcal{P}_i(G)$ which is the worst that can happen in $\mathcal{P}_{i-1}(G)$.

Therefore, by extension:

$$\frac{w(\mathcal{P}_i(G))}{w^j_{\min}} \geq w(\mathcal{P}_{i-j}(G)) \qquad (1 \leq j \leq i)$$

Thus we have:

$$\sum_{i=0}^{n-1} w(\mathcal{P}_i(G)) \leq \sum_{j=0}^{n-1} \frac{w(\mathcal{P}_n(G))}{w^j_{\min}}$$

This leads to two possible cases, namely $|1/w_{\min}| \neq 1$, and $w_{\min} = 1$. We must then define the bound based on $w_{\min}$.

If $|1/w_{\min}| \neq 1$, letting $r = 1/w_{\min}$ we see:

$$\begin{aligned}\sum_{j=0}^{n-1} r^j &= \frac{1 - r^{n+1}}{1 - r} \\ &= \frac{1 - (1/w_{\min}^{n+1})}{1 - (1/w_{\min})} \\ &= \frac{w_{\min}^{n+1} - 1}{w_{\min}^{n+1} - w_{\min}^n}\end{aligned}$$

Otherwise, $w_{\min} = 1$, and the sum is merely n.

Therefore, defining:

$$f(w_{\min}) = \begin{cases} n & \text{if } w_{\min} = 1 \\ \frac{w_{\min}^{n+1} - 1}{w_{\min}^{n+1} - w_{\min}^n} & \text{otherwise} \end{cases}$$

we have:

$$\sum_{i=0}^{n-1} w(\mathcal{P}_i) \leq f(w_{\min}) w(\mathcal{P}_i)$$

We can now use the same argument as in the unweighted case to argue that no edge connecting states in level i to $i+1$ for $0 \leq i < n$ is overloaded. First we note that:

$$w(x)w(y) \leq \left(\frac{w_{\max}}{w_{\min}}\right)^n w(p)w(y)$$

This is due to the fact that the maximum $w(x)$ can be is $w_{\max}^n$ (*i.e.* a path of maximum length with all maximum edges), and the minimum $w(p)$ can be is $w_{\min}^n$ (*i.e.* a path of maximum length will all minimum edges). Since $w(\cdot)$ is proportional to $\pi(\cdot)$, and each edge e is used at most once, we can now bound the congestion as follows:

$$\begin{aligned}\frac{1}{\pi(p)P(p,p')} \sum_{\gamma_{xy} \ni e} \pi(x)\pi(y)|\gamma_{xy}| &\leq \left(\frac{w_{\max}}{w_{\min}}\right)^n \frac{n\pi(p)\pi(y)}{\pi(p)\frac{1}{2n^2} \cdot \min\{w_{\min}, 1/w_{\max}, 1\}} \\ &\leq \left(\frac{w_{\max}}{w_{\min}}\right)^n 2n^3 \max\{w_{\max}, 1/w_{\min}, 1\}\pi(y) \\ &\leq \left(\frac{w_{\max}}{w_{\min}}\right)^n \frac{2n^3 \max\{w_{\max}, 1/w_{\min}, 1\}w(y)}{w(\mathcal{P}(G))}\end{aligned}$$

We note that $w(\mathcal{P}(G))$ will dominate the entire expression. We will therefore see that the congestion which we will define over the canonical path between states representing paths of maximal length will be larger, and thus this congestion can be ignored. The factor to consider, however, will be $f(w_{\min})$, which will defines how the set of paths of length less than n maps into the set of paths of length n.

We now define $cp(e) = \{(x, y) : \gamma_{xy} \ni e\}$ the set of all canonical paths which go through e. We can then define the injective function $\eta_e : cp(e) \to \Omega$ as in the unweighted case. Doing so ensures that η_e is injective.

In the unweighted case it was easy to show that $\pi(p)\pi(\eta_e(x, y)) = \pi(x)\pi(y)$. In the weighted case we now must show that $w(x)w(y) \lessapprox w(p)w(\eta_e(x, y))$. That is, we must show that $w(x)w(y)$ is greater than $w(p)w(\eta_e(x, y))$ by at most a polynomial factor. Unlike in the simple bit string example, this proves to be much harder. This is due to the fact that $\eta_e(x, y)$ does not (necessarily) preserve any order of vertices. Thus when trying to show proportionality of weights, $\eta_e(x, y)$ will be essentially random. In some cases this will result in the introduction of "bad" edges into the canonical path (i.e. edges of poor weight like $w_{\min}$), when both x and y contain only high weight edges (i.e. only of weight $w_{\max}$). This means that a potential bottleneck exists in the Markov chain causing it to be difficult to get from one highly likely configuration to another. In general this will result in a higher mixing time, as in this case.

First we must make a note of how each step along the canonical path changes p and p'. Since we start at x, initially $p = x$ and thus $w(p) = w(x)$. After $|x|/2$ steps, p has at least its first $|x|/2$ vertices in common with y. Thus each step after that point will bring $w(p)$ closer to $w(y)$. With this in mind, it is easy to see that $w(p)$ will be most dissimilar from $w(x)$ and $w(y)$ in the first $|x|/2$ steps. In general, we know that the first i vertices in p will be in common with y. It is easy to see that pulling the first vertex to the front of the path will create at most two new links; namely the link between the new first vertex and the old first vertex, and the link between the two vertices which used to be connected to the new first vertex. For example, if vertex 3 is to be the new first vertex in the path: 123456, we get

312456 as the first step in the canonical path. In this path, the link between 3 and 1, and the link between 2 and 4 need not be present in either x or y. Each additional edge which is "pulled" to the front, however, will only add 1 new potential "bad" edge, which will be the edge created by it vacating its original position. This is due to the fact that there is now a "buffer" zone between the first i vertices in p, and the original first vertex (i.e. the first vertex of x) where a potential bad edge already may exist. Thus if in our example 5 is the next vertex in the final path, we get: 351246. The "new" potential bad edge connecting 5 to 1 now supplants the "old" potential bad edge connecting 3 to 1. In this way, we see that at each step after the first only 1 new potential bad edge arises.

As we have seen, at step $|x|/2$ at least half of the edges agree with y. It is easy to see that the above argument causes maximum difference at step $|x|/2 - 1$, leaving at most $|x|/2 - 1 + 1$ new "bad" edges which don't necessarily appear in either x or y, potentially skewing the weights.

With this in mind, and the fact that $\eta_e(x, y)$ can have arbitrarily many bad edges, we see that:

$$w(x)w(y) \leq \left(\frac{w_{\max}}{w_{\min}}\right)^{|\eta_e(x,y)|+|x|/2} w(p)w(\eta_e(x, y)) \tag{4.12}$$

In other words, all of the potential "bad" edges in the right hand side were replaced by a maximally weighted edge.

By (4.12) and the fact that $\pi(\cdot) \propto w(\cdot)$, we have that:

$$\pi(x)\pi(y) \leq \left(\frac{w_{\max}}{w_{\min}}\right)^{|\eta_e(x,y)|+|x|/2} \pi(p)\pi(\eta_e(x, y))$$

Using this and fact that η_e is injective, we can now evaluate $\overline{\rho}$.

$$\begin{aligned}
\frac{1}{\pi(p)P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(x)\pi(y)|\gamma_{xy}| &\leq \frac{1}{P(p,p')}\sum_{\gamma_{xy}\ni e}\left(\frac{w_{\max}}{w_{\min}}\right)^{|\eta_e(x,y)|+\frac{|x|}{2}}\pi(\eta_e(x,y))|\gamma_{xy}| \\
&\leq \left(\frac{w_{\max}}{w_{\min}}\right)^{|\eta_e(x,y)|+|x|/2}\frac{n}{P(p,p')}\sum_{\gamma_{xy}\ni e}\pi(\eta_e(x,y)) \\
&\leq \left(\frac{w_{\max}}{w_{\min}}\right)^{3n/2}\frac{n}{P(p,p')} \\
&\leq \left(\frac{w_{\max}}{w_{\min}}\right)^{3n/2}\frac{n}{\frac{1}{2|p|^2}} \\
&\leq \left(\frac{w_{\max}}{w_{\min}}\right)^{3n/2}2n^3
\end{aligned}$$

Between states of maximal length we have that $\sum_{\gamma_{xy}\ni e}\pi(\eta_e(x,y)) \leq 1$ since π is a probability distribution (and thus the sum over all elements is 1), $|\gamma_{xy}| \leq n$ by definition, and $\eta_e(x,y)$ is injective (and thus relates to, at most, all of the elements in π).

In the unweighted case the congestion was increased by a factor of $16 \cdot 3$ since the canonical path between maximally length paths actually defined the canonical path between 16 different pairs of states. In the weighted case, there are $f(w_{\min})^2$ states which use the same canonical path, and thus the congestion will be increased by a factor of $3f(w_{\min})^2$. (As in the unweighted case, the factor of 3 must be included due to the length of a canonical path once again increasing from at most n, to at most $3n$).

Therefore, $\overline{\rho} \leq f(w_{\min})^2\left(\frac{w_{\max}}{w_{\min}}\right)^{3n/2}6n^3$, and is polynomial in n as long as $\left(\frac{w_{\max}}{w_{\min}}\right)^{3n/2}$ is reasonably close to 1.

We finally consider $\pi(x)$. Since there are at most $4n!$ paths in G, we see that $w(\mathcal{P}(G)) \leq 4n!w_{\max}^n$. Thus the minimum probability of any path is $\min\{1/(4n!w_{\max}^n), w_{\min}^n/4n!w_{\max}^n\}$. The claimed bound follows as this is how $g(w_{\max}, w_{\min}, n)$ was defined. □

4.2.3 FPRAS for the Weighted Complete Graph

Given the previous theorems for the weighted complete graph, the following follows almost immediately.

Theorem 4.2.3. *There exists an FPRAS for the weighted complete graph on n vertices which runs in time bounded by:* $672(n+1)n^5\epsilon^{-2}f(w_{\min})^2\left(\frac{w_{\max}}{w_{\min}}\right)^{3n/2}(2\ln(3n(n+1)/\epsilon)+\ln(1/g(w_{\max},w_{\min},n)))$.

Proof of Theorem 4.2.3. Combining Theorem 4.2.1 with Theorem 4.2.2, the claimed bound is immediate.

□

Before concluding this section, we note that Theorem 4.2.3 gives a polynomial running time bound if $w_{\max}/w_{\min} \leq 1 + c\log(n)/n$ for some constant c.

4.3 Unweighted Tree

The proof of the existence of an FPRAS for counting the number of paths in an unweighted tree will proceed similarly to the previous cases.

4.3.1 *Approximate Counting $\leq$ Almost Uniform Sampling*

Theorem 4.3.1. *Given a tree $T = (V, E)$ where $|V| = n$, and $|E| = m$, and a FPAUS $\mathcal{S}$ with running time bounded by $T(n, \delta)$, there exists a FPRAS for counting the number of paths in T ($|\mathcal{P}(T)|$) which runs in time $75n^2\epsilon^{-2}T(n, \epsilon/6k)$.*

Proof of Theorem 4.3.1. Let $\mathcal{S}$ be the almost uniform sampler from the theorem, then the approximation scheme proceeds as follows. Let $T = (V, E)$ be a tree such that $V = \{v_1, v_2, \ldots, v_n\}$, ordered such that removing any number of vertices from the end of the list results in a vertex induced subgraph that is still a tree. T_i is thus the vertex induced subgraph resulting from the first i vertices $(v_0, v_1, \ldots v_i)$ in T, $0 < i \leq n$. Therefore, given T_i, T_{i-1} results from the deletion of the vertex v_i. We can then approximate $|\mathcal{P}(T)|$ as a product of ratios:

$$|\mathcal{P}(T)| = (\varrho_1\varrho_2\cdots\varrho_m)^{-1}$$

where:

$$\varrho_i = \frac{|\mathcal{P}(T_{i-1})|}{|\mathcal{P}(T_i)|}$$

(Note that $|\mathcal{P}(T_0)| = 1$ since it contains the trivial path of no vertices.) Since T_i contains all of the paths in T_{i-1}, $\mathcal{P}(T_{i-1}) \subseteq \mathcal{P}(T_i)$. Also, $\mathcal{P}(T_i) \setminus \mathcal{P}(T_{i-1})$ can be mapped into $\mathcal{P}(T_{i-1})$. This can be done by simply removing vertex $\{v_i\}$ from the paths in $\mathcal{P}(T_i) \setminus \mathcal{P}(T_{i-1})$ (i.e. send all of the paths to $\mathcal{P} \setminus \{v_i\}$). Since that vertex can occur at an endpoint in a string (since it is a leaf node as per the definition), there will be at most 1 path mapped to each $\mathcal{P} \setminus \{v_i\}$. Hence,

$$\frac{1}{2} \leq \varrho_i \leq 1. \tag{4.13}$$

We now note that these are exactly the bounds which were achieved in the binary string example. Using the resulting computations, we have a runtime for the algorithm which is $75n^2\epsilon^{-2}T(n, \epsilon/6k)$. □

4.3.2 Existence of an FPAUS

Given that we have an FPRAS which runs in polynomial time, we now define an FPAUS as follows.

Theorem 4.3.2. *There exists an FPAUS for sampling paths from the unweighted complete graph $G = (V, E)$ with run time bounded by:* $8n^2(n^2+1)(2\ln(\epsilon) + \ln(n^2+1))$.

Proof of Theorem 4.3.2. Let $M_T = \{N_t : t \in \mathbb{N}_0\}$ be a Markov chain such that N_i is a path in tree $T = (V, G)$ where $|V| = n$, and thus the state space Ω is all of the paths in T. We then define the transitions from each state $x = (x_1 \dots x_j) \in \Omega$ to $y = (y_1 \dots y_{j'}) \in \Omega$ in Algorithm 6.

Informally, the process M_T is a self-avoiding random walk (with stationary moves) on a tree T with n vertices. To transition between states, a vertex v is chosen u.a.r. from the set of vertices, and either the head or tail is chosen. If the vertex v is the head (tail), then

UNWEIGHTED-TREE-TRANSITION(x)

With probability $\frac{1}{2}$ let $y = x$; otherwise
Select v u.a.r. from V
Select a u.a.r. from $\{0, 1\}$
if $|x| = 0$
  then $y \leftarrow (v)$
elseif $a = 0$
  then if $v = x_1$
    then $y \leftarrow (x_1 \dots x_j)$
  elseif x_0 is adjacent to v, and $v \notin x$
    then $y \leftarrow (v x_1 \dots x_j)$
  else
    $y = x$
elseif $a = 1$
  then if $v = x_j$
    then $y \leftarrow (x_1 \dots x_{j-1})$
  elseif x_j is adjacent to v, and $v \notin x$
    then $y \leftarrow (x_1 \dots x_j v)$
  else
    $y \leftarrow x$

Algorithm 6: *Markov chain which defines how to transition between paths in the unweighted tree.*

the new path is the path without that vertex. Otherwise, if the head (tail) is adjacent to the chosen vertex v, and v isn't already in the path, then v is added to the path.

The Markov chain M_T is ergodic since all states communicate, and the loops ensure aperiodicity. Since the probability of moving from any state x to any other state y is the same as moving from state y to state x, the Markov chain is symmetric, and therefore the stationary distribution is uniform over Ω.

Next we will set up a canonical path γ_{xy} between states $x = (x_1 x_2 \dots x_i)$ and $y = (y_1 y_2 \dots y_j)$. The canonical path γ_{xy} from x to y is defined by (at most) $2n$ edges e_1 to e_{2n}, where edge e_k is defined as:

$$e_k = \begin{cases} ((x_1 \dots x_{i-k}), (x_1 \dots x_{i-k-1})) & \text{if } 0 \leq k < i \\ ((y_1 \dots y_{k-i}), (y_1 \dots y_{k-i+1})) & \text{if } i \leq k < j + i \end{cases}$$

Therefore, the canonical path proceeds in two stages. Stage one consists of shortening the

starting path to the empty path by removing vertices from the end. The second stage consists of extending the empty path to the final path by appending vertices to the end.

We will then consider an oriented edge e, and then define $cp(e) = \{(x, y) : \gamma_{xy} \ni e\}$ the set of all canonical paths which go through e. In previous proofs we would then define the function $\eta_e : cp(e) \to \Omega$ to be injective in order to compute $\overline{\rho}$. In this case, however, this is not necessary. Instead, we will define $\eta_e(x, y) = x$.

With this definition of η_e, we now evaluate $\overline{\rho}$. Note that: $\pi(x)\pi(y) = \pi(p)\pi(\eta_e(x, y))$ (since the stationary distribution is uniform), and therefore we have:

$$\begin{aligned}
\frac{1}{\pi(p)P(p,p')} \sum_{\gamma_{xy} \ni e} \pi(x)\pi(y)|\gamma_{xy}| &= \frac{1}{\pi(p)P(p,p')} \sum_{\gamma_{xy} \ni e} \pi(p)\pi(\eta_e(x,y))|\gamma_{xy}| \\
&\leq \frac{2n}{P(p,p')} \sum_{\gamma_{xy} \ni e} \pi(\eta_e(x,y)) \\
&\leq \frac{2n}{P(p,p')} \sum_{q=1}^{(n^2+1)^2} \frac{1}{n^2+1} \\
&\leq \frac{2n}{\frac{1}{2n}}(n^2+1) \\
&\leq 4n^2(n^2+1)
\end{aligned}$$

We have that $|\gamma_{xy}| \leq 2n$ by definition. It is important to note that $\sum_{\gamma_{xy} \ni e} \pi(\eta_e(x, y)) \leq \sum_1^{(n^2+1)^2} 1/(n^2+1)$ follows from the fact that the value for $|\Omega|$ is known to be $n^2 + 1$. Thus $|cp(e)|$ (*i.e.* the number of canonical paths which use edge e) is at most $(n^2+1)^2$, and $\pi(\cdot) = 1/(n^2+1)$. Therefore we have that $\overline{\rho} \leq 4n^2(n^2+1)$, and the claimed bound is immediate. □

4.3.3 FPRAS for the Unweighted Tree

Given the previous theorems, we can now prove the following.

Theorem 4.3.3. *There exists an FPRAS for counting the number of paths in a tree T with n vertices which runs in time bounded by:* $600n^4(n^2+1)\epsilon^{-2}(2\ln(6k/\epsilon) + \ln(n^2+1))$.

Proof of Theorem 4.3.3. Combining Theorem 4.3.1 with Theorem 4.3.2, the claimed bound is immediate. □

4.4 Weighted Tree

In the previous sections we were able to prove the existence of an FPRAS (with certain restrictions in the case of the weighted complete graph). In this case, however we prove that, with the given underlying Markov chain, the chain is torpidly mixing. To prove rapid mixing in the previous cases we used the canonical path technique. To prove the chain is torpidly mixing, however, we will use the conductance technique outlined in section 3.1.2.

4.4.1 Proof of Torpid Mixing

Theorem 4.4.1. *There exists a family of graphs D_n for which the Markov chain defined by Algorithm 7 is torpidly mixing.*

Proof of Theorem 4.4.1. To prove torpid mixing, we first define the Markov chain on the weighted tree.

That is, this is exactly Algorithm 6 with the Metropolis-Hastings algorithm added.

In order to prove that the mixing time of the Markov chain on this graph is torpidly mixing, we will use the concept of conductance outlined in section 3.1.2. In order to do this, we will use the family of graphs D_n defined below.

Let D_n denote the graph composed of two paths $A = (a_1 a_2 \dots a_{n+1})$ and $B = (b_1 b_2)$ (*i.e.* A is the path of length $n+1$, and B is the path of length 2) with an edge connecting a_1 to b_1. Let edge $e_{AB} = (a_1, b_1)$ have weight $w_{\min} = 1/n^n$, and all of the edges in A and B have weight $w_{max} = 1/w_{\min} = n^n$. Note that the resulting graph is still a tree, and the description of the length of the instance is polynomial in n, because only approximately $n \log n$ bits are required to describe n^n.

As we are using the conductance technique outlined in section 3.1.2, let the set to be considered $S = \{(a_1 \dots a_{n+1})\}$. We now note that $w(a_1 \dots a_{n+1}) = w(b_2 b_1 a_1 \dots a_{n+1})$,

WEIGHTED-TREE-TRANSITION(x)

With probability $\frac{1}{2}$ let $y = x$; otherwise
Select p u.a.r. from V
Select a u.a.r. from $\{0, 1\}$
if $|x| = 0$
  then $z \leftarrow (p)$
elseif $a = 0$
  then if $x_0 = p$
    then $z \leftarrow (x_1 \dots x_j)$
  elseif x_0 is adjacent to p, and $p \notin x$
    then $z \leftarrow (p x_0 \dots x_j)$
  else
    $z \leftarrow x$
elseif $a = 1$
  then if $x_j = p$
    then $z \leftarrow (x_0 \dots x_{j-1})$
  elseif x_j is adjacent to p, and $p \notin x$
    then $z \leftarrow (x_0 \dots x_j p)$
  else
    $z \leftarrow x$
$y \leftarrow z$ with probability $\min\{1, \pi(z)/\pi(x)\}$; otherwise
$y \leftarrow x$.

Algorithm 7: *Markov chain which defines how to transition between paths in the weighted tree.*

and thus $w(S) \leq w(\Omega)/2$. In order to compute ∂S (the probability of leaving set S), the probability that a path in S adds edge e_{AB}, or removes vertices a_1 or a_{n+1} needs to be computed.

We first note that $w(b_1 a_1 a_2 \dots a_n a_{n+1}) = w(a_2 \dots a_n a_{n+1}) = w(a_1 a_2 \dots a_n)$ since in each case we multiply in a factor of $1/n^n$. In the first case this is due to adding an edge of weight $1/n^n$, while in the remaining cases this is due to the removal of an edge with weight n^n. Thus the probability of $(a_1 a_2 \dots a_n a_{n+1})$ transitioning to any of these states is the same, more specifically (recalling that we're trying to determine $\sum_{i \in S, j \notin S} w(i) P(i, j)$):

$$(n^n)^n \left(\frac{1}{n+3} \frac{1}{4} \frac{(n^n)^{n-1}}{(n^n)^n} \right) = \frac{n^{n^2}}{4(n+3)n^n}.$$

This is due to the fact that: $w(a_1 a_2 \dots a_n a_{n+1}) = n^{n^2}$, the probability of choosing vertex b_1,

a_1, or a_{n+1} is $1/(n+3)$ (each), the probability of choosing the appropriate side given the choice of vertex is $1/2$, the probability of avoiding the self loop is $1/2$, and the Metropolis-Hastings filter says to transition only with probability $1/n^n$.

Therefore,

$$\partial S = 3\frac{n^{n^2}}{4(n+3)n^n}$$

We must now compute $w(S)$. Since S only contains one path, $w(S)$ is the weight of the single path, which we have already noted to have weight n^{n^2}. Therefore, the conductance is upper bounded by:

$$\begin{aligned}
\Phi &\leq \frac{\partial S}{S} \\
&\leq \frac{3n^{n^2}/4(n+3)n^n}{n^{n^2}} \\
&\leq \frac{3n^{n^2}}{4(n+3)n^n n^{n^2}} \\
&\leq \frac{3}{4(n+3)n^n}
\end{aligned}$$

Since $w(\partial S)/w(S)$ is inverse exponential, by Theorem 3.1.2 we have that the Markov chain defined by Algorithm 7 is torpidly mixing. □

Chapter 5

Sampling From Counting

In the previous chapter, the concept of almost uniform sampling was used to approximately count the number of paths in a graph. This chapter will show that the converse can also be accomplished; namely if one can approximately count a structure, one can almost uniformly sample from the structure. To this end, an algorithm for counting the number of paths in trees and DAGs (directed acyclic graphs) will be developed. This algorithm will then be used to sample paths from the respective graph (proportional to their weight).

Since the algorithms use the idea of a rooted tree, the links in the tree gain an orientation. Since in this context in both a tree and a DAG there is no way to return to a vertex upon leaving it (otherwise a cycle would exist), both algorithms will treat trees and DAGS the same.

5.1 Counting Paths

While there is only one way to get from any vertex to any vertex in a tree, there are potentially exponentially many ways in a DAG. Thus while one could find the total weight of all of the paths through brute force in polynomial time in a tree, in a DAG it might be exponential. To avoid this, we will devise an efficient approach which uses dynamic programming.

Before proceeding, the following definitions will be useful. Let $G = (V, E)$ be the tree or DAG of interest, and $weightMatrix$ be an $n \times n$ matrix where n is the number

of vertices in V. G is said to be *rooted* at vertex v if all edges are oriented *away* from v[1]. Next define $weight(a, b)$ as the weight of the edge from a to b (or undefined if no such edge exists). Finally, let $children(b, a)$ be a set containing children of b given that the underlying graph is rooted at vertex a.

5.1.1 Algorithm Sketch

For each vertex r, the algorithm first determines the weight of all of the paths which are rooted (i.e. begin) at r. This is accomplished by building a table where entries of the form $weightMatrix[a][b]$ denote the weight of paths rooted at b given that the graph is rooted at a. The weight of all of the paths rooted at b is defined as follows:

$$weightMatrix[a][b] = 1 + \sum_{j \in children(b,a)} weightMatrix[a][j] * weight(b, j)$$

Therefore, for each vertex $v \in V$, $weightMatrix[v][v]$ defines precisely the number of paths which start at v. Therefore the sum of the weights of all of the paths in $\mathcal{P}(G)$ is merely 1 plus [2] the sum of all of the elements on the main diagonal.

5.1.2 Algorithm

WEIGHTED-TREE-PATH-COUNTING()
1 **for** each node in $v \in V$
2 **do** WEIGHTED-ROOTED-TREE-PATH-COUNTING(v, v)

Algorithm 8: *Recursive algorithm driver which determines the sum of the weights of all paths in G*

[1] To achieve this in a tree, merely changes the edges from undirected, to directed away from the root. In a DAG, any edge that does not conform to the definition is not considered.

[2] In order to count the empty path.

WEIGHTED-ROOTED-TREE-PATH-COUNTING$(root, node)$

1 $weightMatrix[root][node] \leftarrow 1$
2 **for** each node $child \in children(node, root)$
3 **do** WEIGHTED-ROOTED-TREE-PATH-COUNTING$(root, child)$
4 $weightMatrix[root][node] \leftarrow weightMatrix[root][node] +$
$weightMatrix[root][child] \cdot weight(node, child)$

Algorithm 9: *Recursive algorithm which determines the sum of the weights of all paths rooted at a vertex.*

5.2 Sampling Paths

Given the above counting algorithm, sampling follows as a result. Once again the algorithm will treat trees and DAGs the same.

5.2.1 Algorithm Sketch

In order to sample almost uniformly, the algorithm first runs the counting algorithm, obtaining $weightMatrix$. It then uses the table to determine a starting vertex, choosing to add vertex v to the sample path s as the starting vertex with probability equal to the weight it adds to the weight of all of the paths in the graph (i.e. $weightMatrix[v][v]$). Subsequently, given that $|s| = k$, vertex $w \in children(s_k, s_1)$ is appended to s with probability $weightMatrix[s_1][w] * weight(s_{k-1}, w) / weightMatrix[s_1][s_k]$. That is, vertex w is added with weight proportional to the weight of the paths that pass through w given the prefix s. With the remaining probability (*i.e.* $1/weightMatrix[s_1][s_k]$), the current path is returned.

Once a leaf is reached (or as aforementioned no vertex is chosen), the path s is returned as the sample path.

5.2.2 Algorithm

In the following algorithm, let s_1 denote the first vertex, and s_k denote the last vertex in the sample path s. The algorithm then proceeds as follows:

```
WEIGHTED-TREE-PATH-SAMPLING()
 1  s ← ()
 2  total ← 1
 3  WEIGHTED-TREE-PATH-COUNTING()
 4  for vertices v ∈ V
 5       do total ← total + weightMatrix[v][v]
 6  Append vertex r to s with probability weightMatrix[r][r]/total
 7  if No vertex is appended with probability 1/total
 8     then return s
 9  while |children(s_k, s_1)| > 0
10       do Append vertex w ∈ children(s_k, s_1) to s with probability
            weightMatrix[s_1][w] * weight(s_k, w)/weightMatrix[s_1][s_k]
11          if No vertex is appended with probability
              1/weightMatrix[s_1][s_k]
12            then return s
13  return s
```

Algorithm 10: *Polynomial time algorithm to sample paths from the weighted trees proportional to their weights.*

Chapter 6

Conclusions and Open Problems

During the course of this thesis we have proven the existence of FPRAS for multiple combinatorial algorithms. Even with the results, however, there are still open problems which deserve attention.

In Theorem 4.4.1 we proved that the Markov chain which was defined for unweighted trees is torpidly mixing in the weighted case. However in section 5.2 we described a polynomial time algorithm for sampling paths from the weighted tree. While this might seem like a contradiction, the existence of the counting algorithm can be explained due to the size of the combinatorial structure. Since the number of paths in trees is polynomial ($n^2 + 1$), they can be enumerated in polynomial time. Therefore the existence of a polynomial time sampling algorithm is not surprising.

While the Markov chain in Theorem 4.4.1 was proven to be torpidly mixing for unconstrained weights, this raises an interesting open problem. Namely, to prove the existence of a rapidly mixing Markov chain for the weighted tree with restrictions on the weights. Intuitively the existence seems likely since there are only a polynomial number of states, however no proof exists as of yet.

Another problem which deserves further consideration is the almost uniform sampler for the weighted complete graph. While the existence of an AUS was proven in Theorem 4.2.2, the result is suboptimal due to the scheme being exponential in the edge weights. Further research into defining a Markov chain which is polynomial in the edge weights, or proving that all Markov chains are torpidly mixing would therefore be of interest.

There is also interesting experimental research which could be undertaken. Namely the AUS proven in Theorem 4.2.2 has very pessimistic bounds. After some initial experiments, it seems that much lower bounds are possible, and with more work into this, one might be able to determine some regularity which leads to a better bound, or even a different Markov chain which is an FPAUS.

Bibliography

[1] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley, New York, 1992.

[2] Ivona Bezáková. *Faster Markov chain Monte Carlo algorithms for the permanent and binary contingency tables*. PhD in Computer Science, The University of Chicago, 2006.

[3] Ivona Bezáková, Daniel Štefankovič, Vijay V. Vazirani, and Eric Vigoda. Accelerating simulated annealing for the permanent and combinatorial counting problems. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 900–907, New York, NY, USA, 2006. ACM.

[4] Béla Bollobás. *Random Graphs*. Cambridge University Press, New York, second edition, 2001.

[5] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335, 1995.

[6] Nick Duffield and Matthias Grossglauser. Trajectory sampling: White paper. Technical report.

[7] Martin E. Dyer and Alan M. Frieze. Computing the volume of convex bodies: A case where randomness provably helps. In *PSAM: Proceedings of the 44th Symposium in Applied Mathematics, American Mathematical Society*, 1991.

[8] Martin E. Dyer, Alan M. Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. In *IEEE Symposium on Foundations of Computer Science*, pages 210–217, 1999.

[9] Zhouyu Fu, Weiming Hu, and Tieniu Tan. Similarity based vehicle trajectory clustering and anomaly detection. *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, 2:II–602–5, Sept. 2005.

[10] Luis Goncalves, Enrico Di Bernardo, and Pietro Perona. *Seeing, Thinking, and Knowing*, chapter Movemes for Modeling Biological Motion Perception, pages 143–170. A. Carsetti ed., Kluwer Academic Publishers, 2004.

[11] Olle Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2007.

[12] W. Keith Hastings. Monte carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[13] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[14] Stefan Janson, Tomasz Łuczak, and Andrzej Ruciński. *Random Graphs*. Wiley-Interscience, New York, 2000.

[15] Mark Jerrum. Counting, sampling and integrating: Algorithms and complexity. Technical report, The University of Edinburgh, Edinburgh, England, 2003.

[16] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. In *ACM Symposium on Theory of Computing*, pages 712–721, 2001.

[17] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[18] Neal Madras and Gordon Slade. *The Self-Avoiding Walk*. Springer, 1996.

[19] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equations of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087–1091, 1953.

[20] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[21] Dana Randall. Counting in lattices: Combinatorial problems from statistical mechanics. Technical Report TR-94-055, University of California: Berkeley, Berkeley, CA, 1994.

[22] Dana Randall. Rapidly mixing Markov chains with applications in computer science and physics. *Computing in Science and Engineering*, 8(2):30–41, 2006.

[23] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, ninth edition, 2006.

[24] Alistair Sinclair. Improved bounds for mixing rates of Markov chains and multicommodity flow. In *LATIN: Latin American Symposium on Theoretical Informatics*, 1992.

[25] Alistair Sinclair. *Algorithms for random generation and counting: a Markov chain approach*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1993.

[26] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

[27] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[28] Eric Vigoda. Markov chain monte carlo methods. `http://www.cc.gatech.edu/~vigoda/MCMC_Course/index.html`, 2006. Lecture notes for a course taught at Georgia Institute of Technoloy.

[29] Douglas B. West. *Introduction to Graph Theory*. Prentice-Hall, Inc., second edition, 2001.